



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**A FRAMEWORK FOR AUTOMATED DIGITAL
FORENSIC REPORTING**

by

Paul F. Farrell Jr.

March 2009

Thesis Advisor:
Second Reader:

Simson Garfinkel
Chris Eagle

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 16-3-2009			2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) 2007-01-01—2009-10-31	
4. TITLE AND SUBTITLE A Framework for Automated Digital Forensic Reporting					5a. CONTRACT NUMBER	
					5b. GRANT NUMBER	
					5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Paul F. Farrell Jr.					5d. PROJECT NUMBER	
					5e. TASK NUMBER	
					5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943					8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)					10. SPONSOR/MONITOR'S ACRONYM(S)	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT Forensic analysis is the science of finding, examining and analyzing evidence in support of law enforcement, regulatory compliance or information gathering. Today, almost all digital forensic analysis is done by humans, requiring dedicated training and consuming man-hours at a considerable rate. As storage sizes increase and digital forensics gain importance in investigations, the backlog of media requiring human analysis has increased as well. This thesis tests today's top-of-the-line commercial and open source forensic tools with the analysis of a purpose-built Windows XP computer system containing two users that engaged in email, chat and web browsing. It presents the results of a pilot user study of the PyFlag forensic tool. Finally, it presents a technique to use software to do a preliminary analysis on media and provide a human readable report to the examiner. The goal of the technique is to perform rapid triaging of media and allow the human examiner to better prioritize man hours towards media with high return on investment.						
15. SUBJECT TERMS Forensic, Domex, Pyflag, Automation						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 109	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (include area code)	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

A FRAMEWORK FOR AUTOMATED DIGITAL FORENSIC REPORTING

Paul F. Farrell Jr.
Lieutenant, United States Navy
B.S., United States Naval Academy, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
March 2009**

Author: Paul F. Farrell Jr.

Approved by: Simson Garfinkel
Thesis Advisor

Chris Eagle
Second Reader

Dr. Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Forensic analysis is the science of finding, examining and analyzing evidence in support of law enforcement, regulatory compliance or information gathering. Today, almost all digital forensic analysis is done by humans, requiring dedicated training and consuming man-hours at a considerable rate. As storage sizes increase and digital forensics gain importance in investigations, the backlog of media requiring human analysis has increased as well. This thesis tests today's top-of-the-line commercial and open source forensic tools with the analysis of a purpose-built Windows XP computer system containing two users that engaged in email, chat and web browsing. It presents the results of a pilot user study of the PyFlag forensic tool. Finally, it presents a technique to use software to do a preliminary analysis on media and provide a human readable report to the examiner. The goal of the technique is to perform rapid triaging of media and allow the human examiner to better prioritize man hours towards media with high return on investment.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

1	Introduction	1
1.1	Motivation	1
1.2	This Research	2
2	Background, Existing Tools and Methodologies	3
2.1	Forensic Methodology	3
2.2	Relevant Forensic Techniques	4
2.3	Identity Resolution	6
2.4	Current Tools	6
3	An Analysis of Today's Batch Reports	11
3.1	Building a Test Image Containing Realistic Data	11
3.2	An Analysis of Current Batch Reports	13
3.3	Default Reports Comparison	18
3.4	A Brief PyFlag User Study	21
4	A Vision for Automated Media Reporting	27
4.1	Automated Ingestion	27
4.2	Automated File Analysis	28
4.3	Automated Reporting	30
4.4	Report Distribution	32
5	PyFlag Implementation	33
5.1	AIM Plugin	34
5.2	Report Plugin.	36
5.3	PyFlag Limitations	39
6	Results on Realistic Data	41
7	A Proposed Framework for Automated Reporting	43
7.1	Requirements.	43
7.2	How We Would Implement It	49
8	Conclusions and Future Work	55

8.1	Conclusions	55
8.2	Future Work	56

Appendices

A	Sample Conceptual Report	59
B	Actual Generated Report	63
C	Sample Adium Log	71
D	Sample Pidgin Log	73
E	Sample AOL Instant Messenger Log	75
F	userreport.py	77
G	domex.py	81
H	IMLogMagic.py	83
	List of References	87
	Referenced Authors	91
	Initial Distribution List	93

List of Figures

2.1	A section of a Scalpel configuration file	9
3.1	The Encase user interface.	15
3.2	Encase Batch Report.	16
3.3	Encase Batch Gallery.	17
3.4	The FTK User Interface.	18
3.5	FTK Batch Gallery.	19
3.6	FTK Batch Emails.	20
3.7	Autopsy user interface.	21
3.8	Autopsy general report.	22
3.9	Autopsy file browsing.	23
3.10	PyFlag Scanning Options	24
3.11	PyFlag File View	25
3.12	PyFlag Image Report	26
4.1	An Automated System State Diagram	28

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

2.1	Forensic tools, their costs and customization ability	8
3.1	Test system that was used to generate realistic data image	11
3.2	Analysis Time of Realistic Image	18
3.3	Tasks performed by the users in the pilot PyFlag user study	26
5.1	PyFlag msn_users table in MySQL	35
5.2	PyFlag msn_session table in MySQL	35
5.3	PyFlag tables in MySQL	38
7.1	Features Table	50
7.2	Seen Features Table	50
7.3	ID Table	50

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

I would like to thank everyone who has helped me while in Monterey. Steve Bassi, Greg Roussas, Kyle Sanders, Jessy Cowan-Sharp and Andrew Slack, thank you all for your support and friendship, it has made my time in Monterey something I will always remember. Professor Chris Eagle, thank you for your patience, your encouragement, and for letting me be a part of your sk3wl. Dr. Simson Garfinkel, without you this thesis would never be finished. I admire your curiosity, your drive, and your dedication and count myself lucky to have the privilege of working with you. Mom, Dad and Allison, thank you for always being there for me, even when we were coasts apart. Finally, to my beautiful wife, thank you for everything, I would not be where or who I am without you.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

1.1 Motivation

The amount of data in the world is increasing. In 2002, about 5 exabytes of new data were created, growing 30% yearly since 1999 [36]. The proliferation of digital information storing devices is incredible, and shows no sign of slowing. With smart-phones, iPods and digital cameras, the average American has more storage on his or her body than the first Cray supercomputers[9]. In 2003, 62 million US households owned computers connected to the Internet[34]. In 2008, one terabyte hard drives are sold for \$100. With cheap storage, broadband Internet connections for sharing, and ease of digital creation thanks to still and video cameras, some researchers warn of the oncoming “Exaflood” of information[33] .

As storage becomes cheaper and personal hard drives become larger, the task of the digital investigator grows geometrically harder. Individuals own more and larger media devices than ever before, and sometimes when a crime is committed, evidence may exist somewhere on one of the devices. It is the job of the digital investigator to find what evidence exists and recover that evidence in a sound manner.

One only has to look to the FBI to realize how important digital forensics has become in serious criminal cases. In 2007, the FBI opened it’s 14th Regional Computer Forensics Laboratory (RCFL) and handled 76,581 pieces of media, up from 59,677 in 2006. Nine of the 14 labs reported a larger backlog at the end of 2007 than at the start, despite the increases in funding and staffing levels. In their 2007 annual report, the RCFLs state that “The capacity of electronic devices continues to increase, examiners must review more and more data. Therefore, even if the number of requests decreases, the workload either remains steady or actually increases in many cases”[17]. That same report emphasizes the importance of computer forensics in high profile cases, such as the Bike Path rapist[17], the foiled Fort Dix terrorist attack, and Operation Remaster, the largest piracy case in U.S. history.

As the search space grows faster than law enforcement's ability to search, there exists an opportunity for software to step in and take some of the load from law enforcement personnel. This thesis argues that the current generation of forensic tools are not up to the task because these tools are designed for manual operation by a trained operator. What is needed, instead, is a new generation of tools that can perform automated analysis and reporting.

1.2 This Research

With the potential mountains of information related to any investigation, the forensic examiner has to prioritize their man hours. In large cases it may be infeasible to exhaustively analyze every piece of media. The media must be prioritized for potential evidence it may contain. If multiple computers are seized, the examiner may have no idea which hard drive contains the evidence they are looking for. This thesis proposes an automated reporting system that will give the examiner a brief summary of the data residing on a piece of media. Important information may include email and instant messaging accounts, digital pictures and movies, as well as external devices registered with a particular operating system such as digital cameras or iPods. The report can also look for particular data of interest and flag it for priority analysis, such as images taken by the same camera associated with child pornography. Once the media in a case is ingested and the reports are generated, the examiner can then prioritize their in-depth analysis towards high probability media.

CHAPTER 2:

Background, Existing Tools and Methodologies

The computer forensic field is very active in both the commercial and research sectors. Commercial tools provide standardized interfaces, storage formats, training, certification and support to the forensics community. Some are formally reviewed, tested and analyzed by the National Institute of Standards and Technology (NIST) Computer Forensic Tool Testing (CFTT) program [28] other tools are used without formal testing, raising potential problems if the results are challenged in court.

2.1 Forensic Methodology

There have been many attempts to create a formal methodology for digital forensics. Such a methodology not only aids in training and practice, but also serves as a guide in future tool development. In 2001, The Digital Forensic Research Workshop (DFRWS) identified a seven step process as identification, preservation, collection, examination, analysis, presentation, and decision[13]. In 2004 the U.S. Department of Justice (DOJ) released *Forensic Examination of Digital Evidence: A Guide for Law Enforcement*. This guide outlines 3 principles for digital forensics:

- Actions taken to secure and collect digital evidence should not affect the integrity of that evidence.
- Persons conducting an examination of digital evidence should be trained for that purpose.
- Activity relating to the seizure, examination, storage, or transfer of digital evidence should be documented, preserved, and available for review[35].

In addition to these principles, the Guide explains the steps necessary for digital forensics as assessment, acquisition, examination, and document and reporting.

In his 2005 PhD dissertation, Carrier describes the concepts as transfer, identification, classification, individualization, association and reconstruction [6]. These separate but similar approaches all share common concepts and generalized steps, namely to obtain the data, to analyze the data and to yield a result from the data.

2.2 Relevant Forensic Techniques

One of the reasons that digital forensics is such a demanding profession is that it requires mastery of many different, highly specialized techniques. An added complication is that most of these techniques are in a constant state of flux given the speed with which the industry changes.

2.2.1 Imaging

One of the first techniques used in a digital forensics investigation is to image, or copy, the media to be examined. Though this seems to be a straightforward step at first, modern Operating Systems (OSs) perform many operations on file systems when connected, such as indexing or journal resolution. Without care, media can be modified, however slightly, and the integrity of the evidence can be compromised. For example, OSs that index files may modify the access times of the files being indexed or the act of mounting a disk may cause data in the disk journal to overwrite other data. A trained investigator is required to have intricate knowledge of different operating system behaviors so that residual data will not inadvertently be overwritten.

Both hardware and software solutions have been created to allow imaging that does not modify the drive. For most OSs, there are procedures to follow to protect the media to be imaged. This involves disabling auto-mounting services and accessing the raw device directly. There are also hardware solutions, in the form of write blockers, that physically prevent the OS from modifying the media. Write blockers are most common for hard drives, and provide several variations on implementation technique.

Once the investigator is assured that the source disk will not be modified, the data must be copied off the source disk for analysis. Once again, this seemingly simple procedure has important details that must be considered. A physical media device is normally made of addressable blocks where data is stored. These blocks can be grouped into multiple partitions per device, with potential gaps between the partitions. Partitions are then formatted into filesystems, with certain blocks containing metadata and control data for the file system. To ensure all information is accurately copied from media, the media must be imaged at the block level. Also, if the original media is damaged or presents input/output errors, the imaging software must account for the error, yet try to recover as much of the data as possible.

Imaging is a complex and important task in the realm of digital forensics. To support law enforcement and industry alike, the National Institute of Standards and Technology (NIST),

has created the Computer Forensic Tool Testing (CFTT) Program. CFTT has set standards, created testing utilities, reviewed and certified disk imagers, file recovery software, hardware and software write blockers[28].

2.2.2 Hashing

To quickly identify a file and to provide authenticity that an image or file was not modified, the forensic community adopted cryptographic hashing. Modern hashing functions use one way cryptographic functions to obtain a hash. The uniqueness of the hash depends on the cryptographic function used. MD5 hashing was developed in 1991 by Ron Rivest [?] and was rapidly adopted by the forensics community. NIST soon decided upon SHA-1 as the federal standard [27], but the forensic community continued to use MD5 in most tools because it was faster and produced a shorter hash. In 2004, MD5 was shown to be insecure by Chinese researchers[38]. This research makes relying on MD5 hashes alone questionable in legal contexts.

NIST collects software to hash for it's National Software Reference Library project. NIST hashes the files collected with both MD5 and SHA-1, and plans additional functions in the future[29]. NIST also plans to distribute hashes of individual file "blocks". These hashes are distributed as the Reference Data Set (RDS). The RDS is distributed in 4 compact disc iso images, with RDS 2.21 covering 47,553,722 known files with 14,563,184 unique hashes.

Since basic hashing provides drastically different results for a 1 bit change in a file, research is underway to provide approximate hashing. State of the art in hashing is Multi-Resolution Similarity Hashing based on context triggered piecewise hashing, using hash similarities to form edit distances between files[31].

2.2.3 Carving

One category of tools in the digital forensic toolkit is called file carvers. These tools allow the scanning of disk blocks that don't belong to current files to find deleted data. Carvers use known header and footer signatures to combine these 'unused' inodes into the original files that were deleted[23]. Carving can recover deleted but not overwritten files as well as temporarily cached files on media. An analysis of carving techniques was performed by Mikus in 2005 [25].

Recent advances in carving allowing fragmented files to be recovered with more accuracy. Garfinkel demonstrated file carving with object validation [18], showing it was possible to

validate whether blocks belonged to certain files as they are carved out, allowing fragmented files to be recovered intelligently. In 2008 Pal took validation further to present Sequential Hypothesis Testing using earlier work in Parallel Unique Path [30]. This allowed largely fragmented files to be recovered as long as a sufficient validation function exists for the filetype. Also in 2008 Cohen described advanced JPEG carving, creating a jpeg validator based off of the open source libjpeg and a distance function to find sudden image changes, indicative of an invalid reconstruction[8].

2.3 Identity Resolution

Resolving individual pieces of data or information to an owner or identity is a significant problem in forensics cases. Where multiple users could exist on a single machine or network capture, the process of data ascription becomes complicated. Two techniques for resolution are heuristic techniques like Jonas's work for IBM[24], and probabilistic machine learning techniques developed by researchers for law enforcement[37]. Data is correlated together from various sources into a database and then pointed to by an entity entry meant to represent a person. As more data is ingested, an entity may gain more pieces of data for greater resolution, or new information may arise that causes an entity to split into separate entities. The pieces of data that an entity owns may represent communication to other entities and social networks may now be formed in the database for further analysis.

2.4 Current Tools

The forensic market has created many opportunities for commercial ventures and popular for open source alternatives. Standalone tools that perform specific functions such as extraction of EXIF data from JPEG, are constantly being developed and distributed in the academic and open source communities. These novel functions are eventually incorporated into larger analysis suites. These suites are typically large GUI-based programs that allow an analyst to explore and search the data on a hard drive.

The remainder of this section discusses the most popular forensic suites.

2.4.1 EnCase

EnCase is a forensic suite sold by Guidance Software, certified by the NIST CFTT, and used by many law enforcement agencies throughout the United States. "EnCase Enterprise is a powerful, network-enabled, multi-platform enterprise investigation solution. It enables imme-

diate response to computer-related incidents and thorough forensic analysis. It also preserves volatile and static data on servers and workstations anywhere on the network, without disrupting operations[22].”

EnCase uses a proprietary file format to store its images[19]. The open source library libewf[2] allows other forensic tools to use these images. EnCase has a complicated interface that requires many steps and operations before actionable intelligence can be returned in an investigation. EnCase provides a basic scripting language to automate common tasks in the suite and scripts are often traded in the user community. This scripting language allows custom carving, extracting and reporting, though the reports are limited to within the EnCase viewer and are not immediately able to cross reference other cases. The source code is closed and users are not able to add their own extensions to the program.

Proper EnCase usage, especially in a law enforcement environment, often requires advanced training, which can cost up to \$5,000 for a one-week course[21]. Guidance Software made \$35 million from service and maintenance related income in 2007, almost 44 percent of its overall revenue[20].

2.4.2 FTK

Forensics Tool Kit (FTK) is a forensics suite sold by AccessData. FTK is another commercial tool with a steep learning curve for its users. Like EnCase, FTK has been validated in courtrooms with legal precedent [1]. FTK provides more data rich reports in the default interface than EnCase. FTK does not provide a scripting language and does not allow users to add additional functionality.

2.4.3 Sleuthkit

The world of open source forensics is dominated by The SleuthKit (TSK)[5], the primary tool for extracting files from disk images. TSK is an open source suite of digital forensic tools based on the original The Coroner’s ToolKit tool set[15]. TSK supports file system browsing, string searching, timeline building, and other reports. TSK also has a programming library, allowing other programs to be written on top of it.

Although SleuthKit can be run from the command line, many practitioners find it easier to use a graphical user interface. Two such graphical interfaces are Autopsy[4] and PTK[11]. Both

Name	Scripting	Cost
EnCase	Yes	\$2,708.31
FTK	No	\$2,557.44
Sleuthkit	Yes	Free
PyFlag	Yes	Free

Table 2.1: Forensic tools, their costs and customization ability

Autopsy and PTK are little more than graphical shells which run the TSK commands as child processes and present the results in a web browser for easier visualization than the command line tools.

2.4.4 PyFlag

The Australian government has released the Python Forensic and Log Analysis GUI (PyFlag). PyFlag is an open source forensics suite designed for media and network analysis. PyFlag imports a case image into a back-end database where persistent information is stored for access through a web browser from a client workstation[7]. In practice the database can be on the same system as the client, allowing for a mobile deployment, or on a central server, allowing many investigators to work n the same case at the same time. PyFlag uses TSK for underlying image access and builds individual file analysis, extraction and reporting on top of TSK[7]. PyFlag provides its own scripting language called PyFlash and also allows users to write their own extensions to the suite in python. PyFlag has proven its usefulness by being the main tool to solve the DFRWS 2007 and 2008 forensic challenges.[14].

2.4.5 Stand Alone Carvers

In addition to the tools above, there are several stand alone file carving utilities. Many require compilation and linking to an array of libraries. Very few are maintained in the popular repositories for easy installation. System specific compilation and configuration options are common, and almost all run solely on the command line. The most popular open source carving tools are scalpel by Kornblum[23] and its derivative foremost by Richard[26]. Also available are photorec by CGSecurity and Magic Rescue by Jensen. Scalpel and Magic Rescue rely on ‘magic bits’ of files, often referred to as headers, to identify files for recovery. (For example, Figure 2.1 shows the complexity of the Scalpel configuration file.) While training is normally available free online and users can interact with the developers via email and IRC, the average user can quickly become lost in command flags and version skew among the varied tools.


```

# AOL ART files
#     art      y      150000  \x4a\x47\x04\x0e      \xcf\x7\xcb
#     art      y      150000  \x4a\x47\x03\x0e
#     \xd0\xcb\x00\x00
#
# GIF and JPG files (very common)
#     gif      y      5000000      \x47\x49\x46\x38\x37\x61
#     \x00\x3b
#     gif      y      5000000      \x47\x49\x46\x38\x39\x61
#     \x00\x3b
#     jpg      y      200000000      \xff\xd8\xff\xe0\x00\x10
#     \xff\xd9
#
# PNG
#     png      y      200000000      \x50\x4e\x47?
#     \xff\xfc\xfd\xfe
#
# BMP (used by MSWindows, use only if you have reason to think there
#     are
#     BMP files worth digging for. This often kicks back a lot of
#     false
#     positives
#
#     bmp      y      100000  BM??\x00\x00\x00
#
# TIFF
#     tif      y      200000000      \x49\x49\x2a\x00
#     tif      y      200000000      \x4D\x4D\x00\x2A
#

```

Figure 2.1: A section of a Scalpel configuration file

Recently released is Adroit Photo Recovery[12] using the Parallel Unique Path algorithm discussed earlier; this tool is focused solely on digital picture recovery. This carving tool is both extremely powerful and quite easy-to-use. It is likely to have a significant impact on both the forensic and the data recovery market.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:

An Analysis of Today's Batch Reports

This chapter presents the batch forensic reports created by today's top-of-the-line commercial and open source forensic tools. To test these tools with data that could be published here we built a clean computer system running Windows XP in a VMWare Virtual machine, populated the machine with two user accounts and a standard set of software, and then used the accounts to engage in simulated real-world communications. We also conducted a pilot user study of PyFlag.

The analysis here confirms what many forensic practitioners already know: significant training is needed to use today's forensic tools. The tools have many reports, each one highly specific with a lot of information which easily overwhelms users and customers alike, delaying the finding and use of actionable intelligence. Each report that comes with EnCase internalizes its information, making sharing with other reports difficult, and FTK has no way to script it's multiple reports together into the single report desired.

3.1 Building a Test Image Containing Realistic Data

To evaluate these tools against each other for ease of use and value of automated reports, we created a test image populated with realistic data.

The test image is a VMWare Fusion virtual machine running Microsoft Windows XP Service Pack 3 with an assortment of messaging and communication applications installed as seen in Table 3.1.

Software Package	Version
Windows XP	Service Pack 3
Internet Explorer	6.0.2900.5512.xpsp.080413-2111
Mozilla Firefox	3.0.3
Google Chrome	0.3.154.9
AOL Instant Messenger	6.8.12.4
Pidgin (www.pidgin.im)	2.5.2
Mozilla Thunderbird	2.0.0.17

Table 3.1: Test system that was used to generate realistic data image

3.1.1 Building the Test Image

In our hypothetical scenario, a computer is discovered or acquired and the analyst needs to determine who was using the drive, what they were doing with it, and with whom they were communicating. To enable testing of this scenario, we created a test image with realistic but fictional data.

The image has two users, `domexuser1` and `domexuser2`. (The user accounts were created from the default administrator account and labeled `domexuser1` and `domexuser2` for simplicity.) Each account was created as a standard user with no administrator privileges.

On a separate computer, a `domexuser3` account was created for third party communications, a requirement for portraying a communications network in which there are unknown or not yet analyzed identities. We created a hotmail and gmail account for each user account, for a total of 6 email accounts.

After the user accounts were created, the administrator account was used to install common communication and productivity software, including Microsoft Office 2008, Mozilla Firefox, Mozilla Thunderbird, AOL Instant Messenger, and Pidgin were installed for all users. The installation files were deleted and the recycle bin was emptied, and the administrator account logged off.

Over a course of several days, an experimenter playing the role of one user and then the second exchanged instant messages and emails with `domexuser3`, which resided on a separate system. The two accounts received, edited and saved office document files as well as various media files. Some of these files were then deleted. Email and instant messenger conversations were saved locally on the system. The accounts also visited web pages for news and webmail. Forum and message board accounts were created and used.

`Domexuser1` was the first to log in, and immediately went online to create communication paths. Using Mozilla Firefox, `domexuser1` created a hotmail account, a gmail account, a gchat account and an AOL instant messaging (IM) account, all with the name `domexuser1`. Thunderbird was configured for the accounts and left open to synchronize. AOL instant messenger was connected to the internet and `domexuser1` setup was completed.

The Switch User option in the XP menu was then used to log in as `domexuser2` without logging out `domexuser1`. This time, using Internet Explorer, the same online accounts were

created for the username `domexuser2`. Outlook Express was configured and synchronized for the mail accounts and Pidgin was logged on for the AOL IM and gchat accounts.

Switching between the three accounts, test emails and instant messages were sent back and forth to establish that all were participating. `Domexuser1` then created three Microsoft Word documents and three Microsoft Excel Documents. One document of each type was sent to the email threads, one of each was deleted, and one of each was left alone.

The VMWare image then needed to be imaged for analysis. VMWare Fusion allows the vmdk files to be mounted within userspace. The unix command `dd` was then used with the options `noerror` and `sync` to provide a raw copy of the image. The final image was 40 Gigabytes in size, with much of that space empty.

3.1.2 Why a Test Image

This thesis does not use real data, instead, we created a realistic test image to provide a standardized image to test current tools on and to develop PyFlag modules against. Using realistic data avoids the man hours required to scrub the dataset and results of Personally Identifiable Information (PII). Finally, the use of realistic communications data would require consent of all communicating parties, which was deemed to be too high a hurdle for a single research project such as this.

3.2 An Analysis of Current Batch Reports

All of the reports that were analyzed for this thesis have a similar flaw: the reports are filled with *data*, but they do not present *intelligence*.

The forensic tools analyzed here are able to recover tremendous amounts of raw data, files, streams and associated metadata. This data by itself yields little value in an investigation. There are pages and pages of file names, time stamps, attributes, and other low-level details. None of this data is presented with any higher level of summarization or analysis.

Data must be analyzed in the context of the current case to yield actionable intelligence. This analysis should occur *before* the tool presents its results back to the user: it should not be the task of the user to try to weed through all of the data and make sense of it. Of course it is risky to take the human out of the loop completely—the tool should *not* hide data from the user, nor be biased towards any type of intelligence conclusion. But some decision must be made, for the

simple fact that all of the information cannot fit on a single page. Today's tools typically order their output alphabetically or chronologically. In most cases this is not the optimum ordering.

Each tool analyzed here presents very different reports containing different kinds of information. Both the reports and the interfaces used to obtain them correspond to the forensic tool's internal program architecture, rather than the way that an analyst would hope a tool to work or a report to be organized. To use these tools, the analyst must learn to think the way the tool's original programmer thought.

Today's forensic tools are diverse and designed for use by digital forensic tool experts. The commercial tools are designed with complex interfaces that are confusing to both inexperienced users and to experts who have merely been trained on other tools. Documentation and tutorials are available for all of these programs, but it is highly specific to each tool. This is not surprising: Commercial training and certification on individual tools is significant source of revenue for tool makers, so there is little incentive to standardize. Both the complexity of the commercial tools and their differences from one another are made clear by comparing the EnCase user interface (Figure 3.1) and the FTK user interface (Figure 3.4).

The purpose of this thesis is to describe a next-generation forensic tool that can be used by a person who is familiar with digital forensic abilities and limitations, but who is not an expert in any technique or tool set. The user will need the tool to produce a report with useful information with a minimum of configuration and interaction. Ideally the tool will be a "one-click" solution, producing a report with minimal involvement. The user can then decide if a piece of media requires deeper analysis by an expert, ideally with a focused search for certain information or file types.

3.2.1 EnCase Batch Report

EnCase data analysis begins by ingesting the data into the EnCase program. Each ingest then must be indexed before a useful report can be generated. This indexing uses a word dictionary that the user can add to and then search the image, creating a list of found words. The reporting tab is then populated with the results of the index. The default report (Figure 3.2) shows a file directory tree, which has far more data than is useful, with little filtering or layout consideration. Though this report could be better organized and searched through with more scripting, the built-in EnCase report provides little intelligence to the investigator. Other default reports are a gallery view of all images found and a timeline view of all files on the media. The reports allow

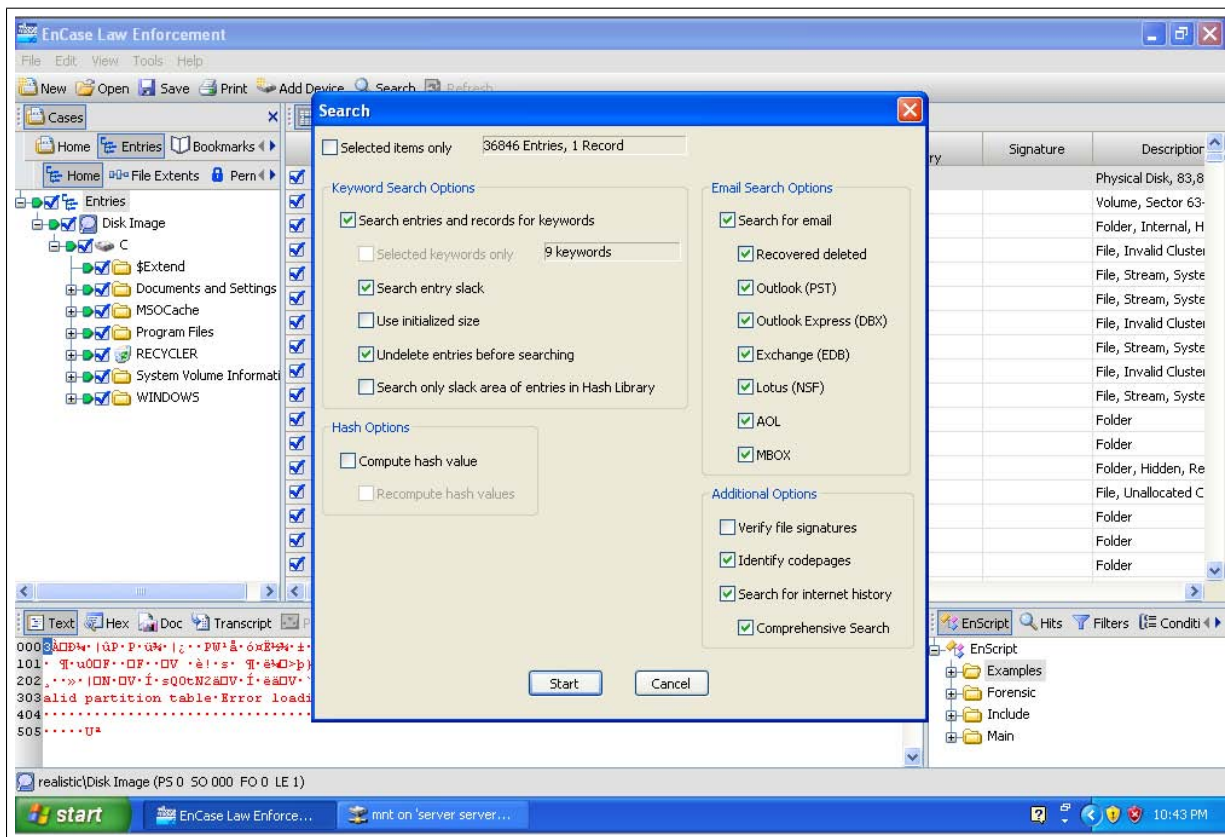


Figure 3.1: The Encase user interface.

filtering by the user according to different file properties. The next step is to process the case, which runs file type specific searches against the media, as well as regular expression searches. EnCase can also mount container files in this process. These reports are then saved as html files in a user chosen directory, and must be viewed by an external web browser.

3.2.2 FTK Batch Report

Like EnCase, FTK must ingest a media file to start analysis. At the ingest phase, FTK allows the user to select how in depth the case index is created, as well as what files are returned in the results. Carving options are also available on ingest, and a list of known types to carve for is presented to the user. In our tests, FTK 2.0 would hang when analyzing files shared across a network, even when the file share was mounted as a drive in windows. We were only able to achieve results with FTK when the data to be analyzed was present on a *local* drive. This is a significant limitation when distributed operations are desired.

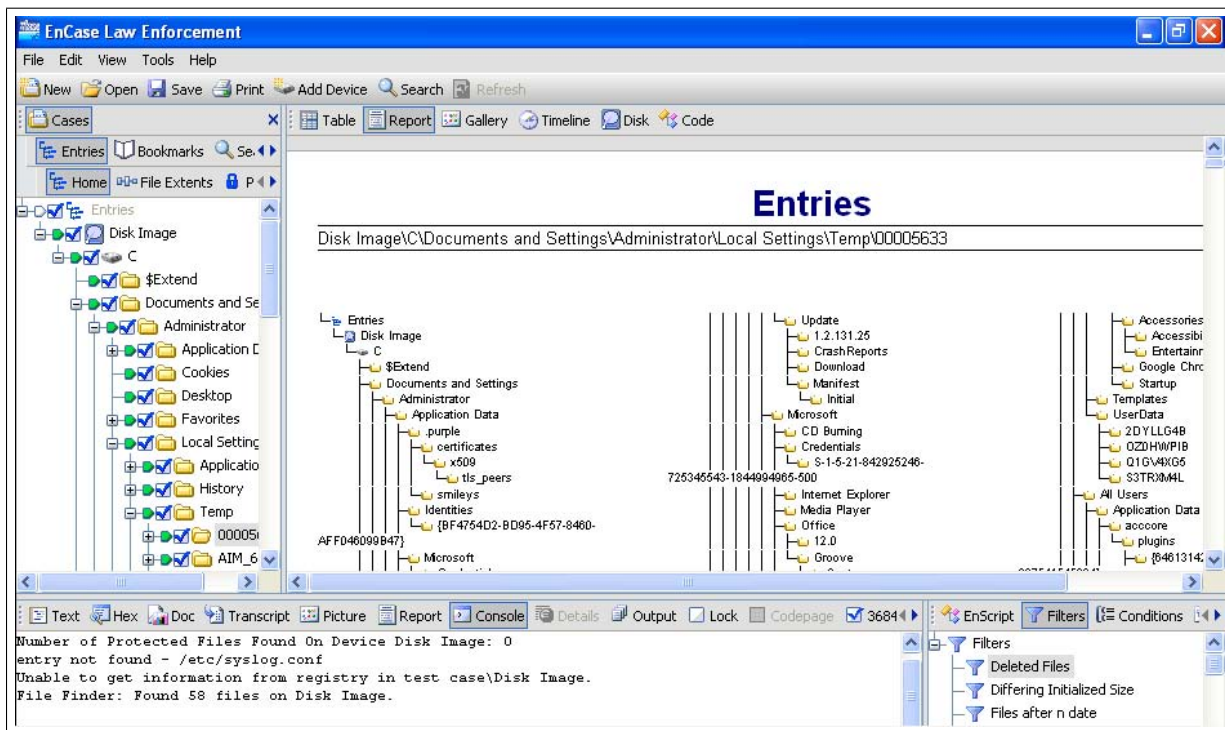


Figure 3.2: Encase Batch Report.

FTK then presents the user several views on the data analyzed. The user can explore the file structure, see all email information, see all graphics, see general file information and statistics, and run searches on the index or the actual media based on regular expressions or words. No additional analysis can be performed, and custom scripts to extract data from known file types or to carve for other file types do not exist. FTK presents the data it finds in an easy to read view, but still presents a *lot* of data to the end user, and the user must search and filter the data for relevant information.

3.2.3 TSK/Autopsy Report

Autopsy ingests do not have the same index requirement the other applications have. Autopsy performs its analysis as the user navigates the web interface, performing the analysis on the fly. However, Autopsy does require precursor steps for timeline related viewing, but only for that mode. Through the web interface, this procedure is described as “process the file system images, collect the temporal data, and save the data to a single file.” Once processed, Autopsy recommends the user view the timeline in a separate application such as a text editor. Autopsy also provides a general report about the ingested media’s metadata. Users can browse the file

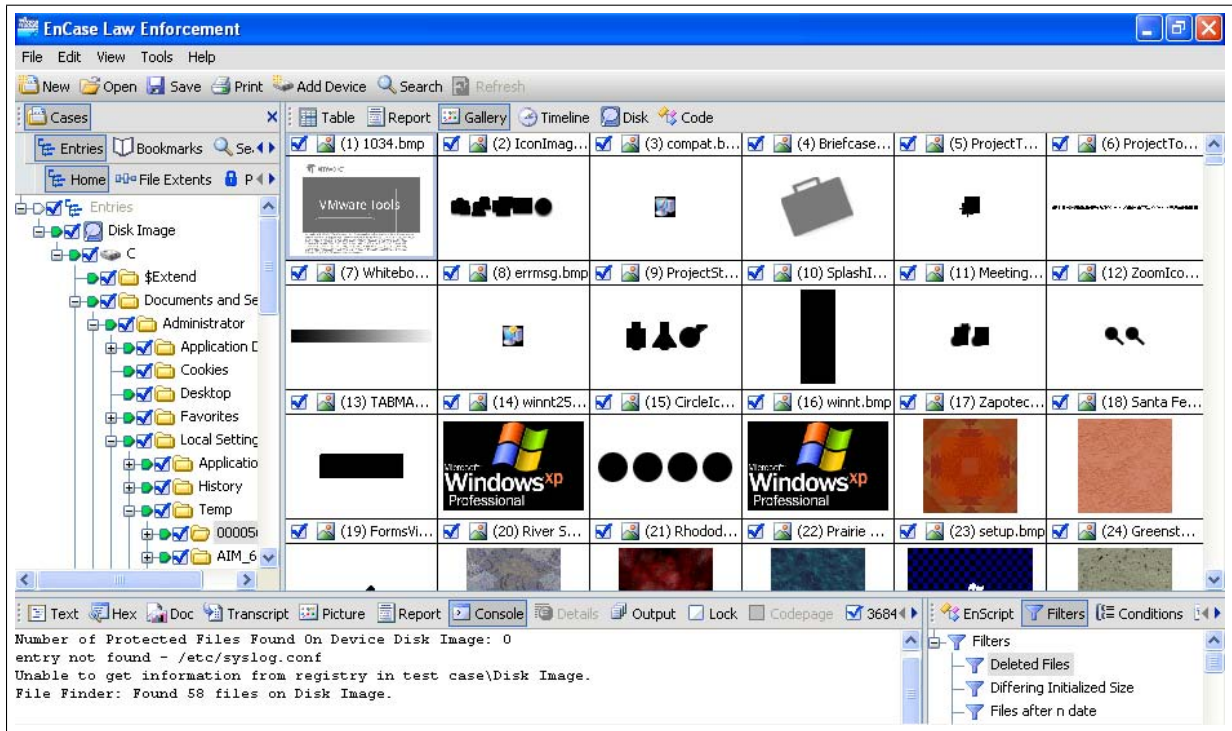


Figure 3.3: Encase Batch Gallery.

structure through autopsy, which issues TSK commands as the requests are made. Autopsy also allows the user to extract all strings from the media image into a file for faster searching, otherwise the media image is searched in real time for the user.

3.2.4 PyFlag Batch Report

PyFlag is a significantly more sophisticated open source forensic tool than Autopsy. Like Autopsy, PyFlag uses SleuthKit for extracting files from disk images, but it then analyzes the files using its own recursive analytical framework. PyFlag is also capable of analyzing intercepted network packets and memory images. All in all, PyFlag is much more comparable to EnCase and FTK than to other open source alternatives.

We ingested the raw image using PyFlag running on an Ubuntu Virtual Machine inside VMWare Fusion on a 2.0 GHz Core 2 Duo Macbook with 4GB of RAM. To test the limits of PyFlag's analysis, all scanners were turned on except for network scanners.

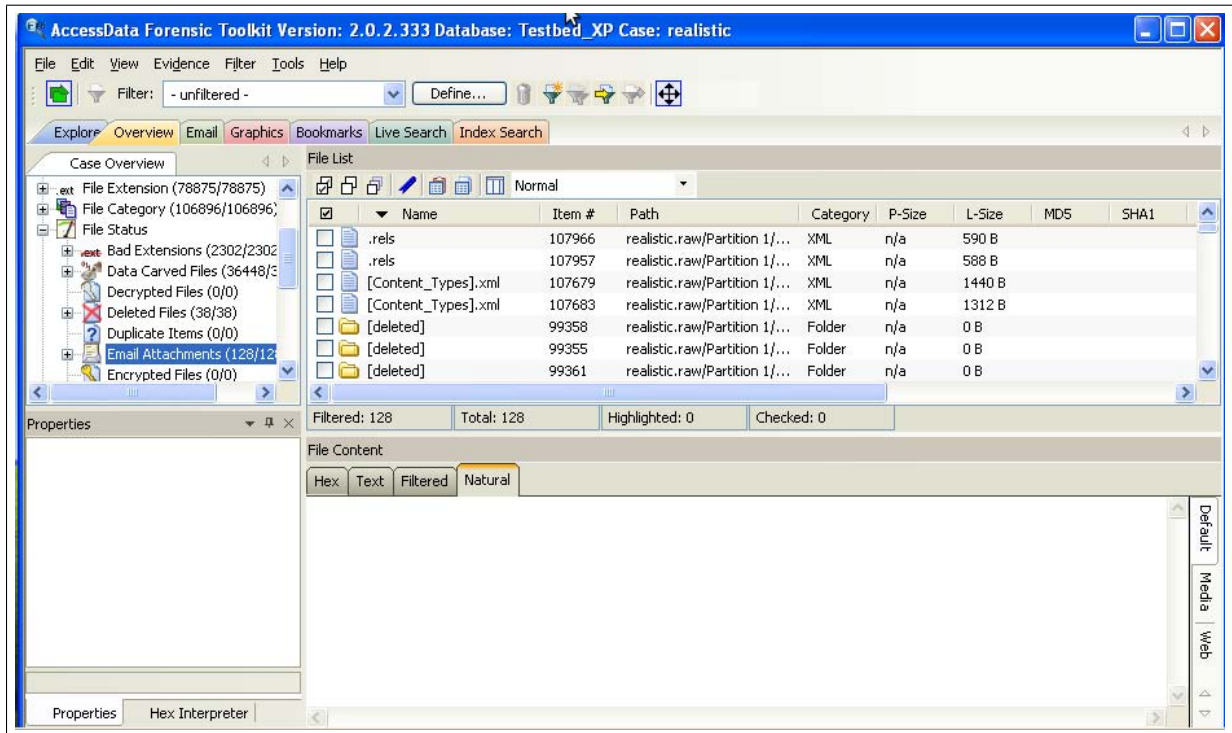


Figure 3.4: The FTK User Interface.

Name	Time
EnCase	48 hours
FTK	30 hours
Sleuthkit/Autopsy	N/A(real-time)
PyFlag	26 hours

Table 3.2: Analysis Time of Realistic Image

Once all the scanners had run and the database was populated, the user could request that the system create a report. At this point the user can create reports from one of three categories: Disk Forensics, Network Forensics, memory Forensics. The most useful batch report for this image were the “I’m Feeling Lucky” reports—a collection of five disk forensics reports. For this image the “File Times” report was also useful; this report was under the top-level “Disk Forensics” category.

3.3 Default Reports Comparison

Each application was run against the 40 GB realistic image previously discussed, to look for information about computer user accounts (domexuser1 and domexuser2), stored com-

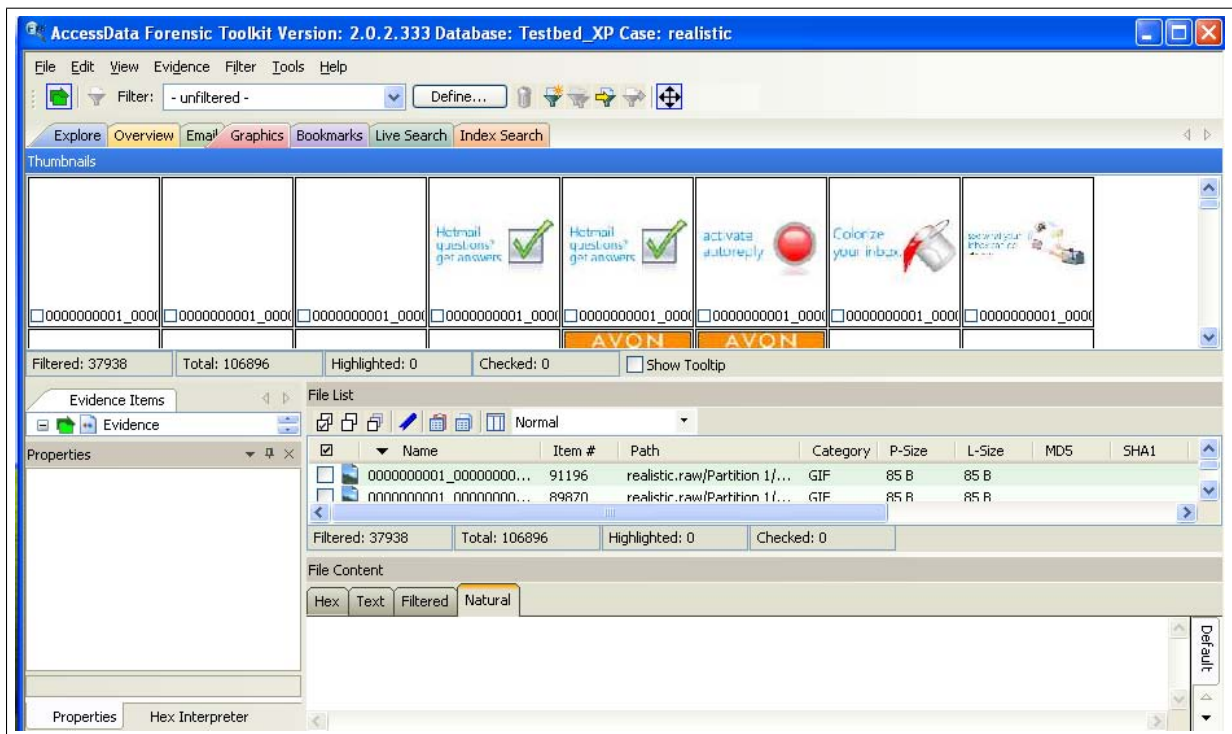


Figure 3.5: FTK Batch Gallery.

munications (email, web and IM logs), and other information (office documents). For each application, the The PyFlag ingestion and reporting option took approximately 24 hours to complete. By re-running all available scanners against the image (only default ones are run at ingestion), analysis took another two hours. The EnCase ingestion, indexing and searching took nearly 24 hours. The Case Processor step of EnCase took another roughly 24 hours as it apparently attempts the most in depth file carving across the image. This carving step The FTK ingestion and analysis took roughly 30 hours. Of note is that each test was run on the same virtual machine hardware, limiting the amount of available memory to 1024MB. On computers with high end hardware, these times may be faster.

The imaged we created to be analyzed has several specific pieces of information that should be presented to the user. Each system has two user accounts in addition to the administrator. Each user account has two email addresses and two instant messenger accounts. Also, domexuser1 has four document files in his My Documents directory and two deleted documents. Each account also communicates with the same third party account via email and instant message, domexuser3.

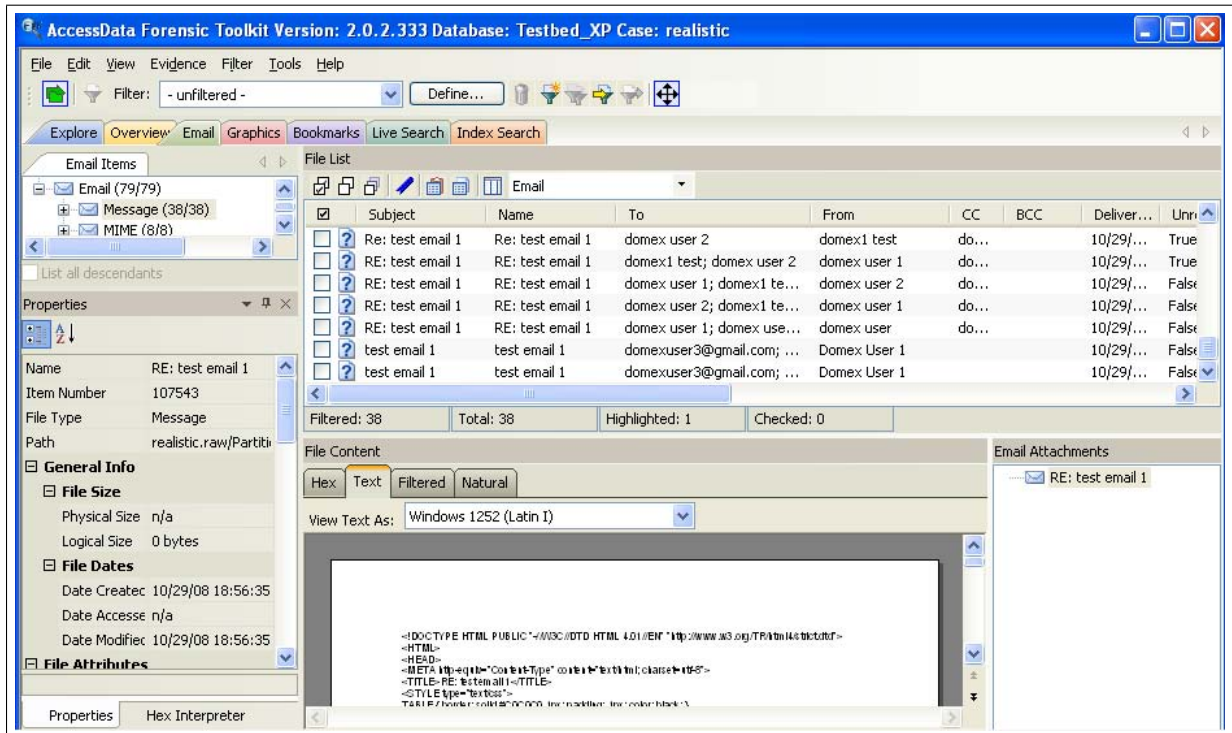


Figure 3.6: FTK Batch Emails.

EnCase allows the user to find and recover all office files, but the user must navigate to the appropriate directories by hand. After the initial 24 hours ingest and analysis, we initially ran the case processor with the entire image selected and all relevant options selected (non-Windows options were disabled.) This analysis took another 24 hours and the report generated had little data and less information. The email accounts and instant messenger communications were not found, even though those options were selected. We re-ran the case processor on only the two user home directories with only the options selected for information we know existed. Even with this narrow search focus, EnCase took almost 12 hours and was unable to report on email or IM communications in the test image.

FTK presented the found email addresses to the user under the email tab. Each email could be seen and could be sorted by to, from, size and date. FTK also allows the user to find files by extension or browse the file structure, recovering all office documents in the image if the user knows where to look. FTK did not present any information about the instant message communications.

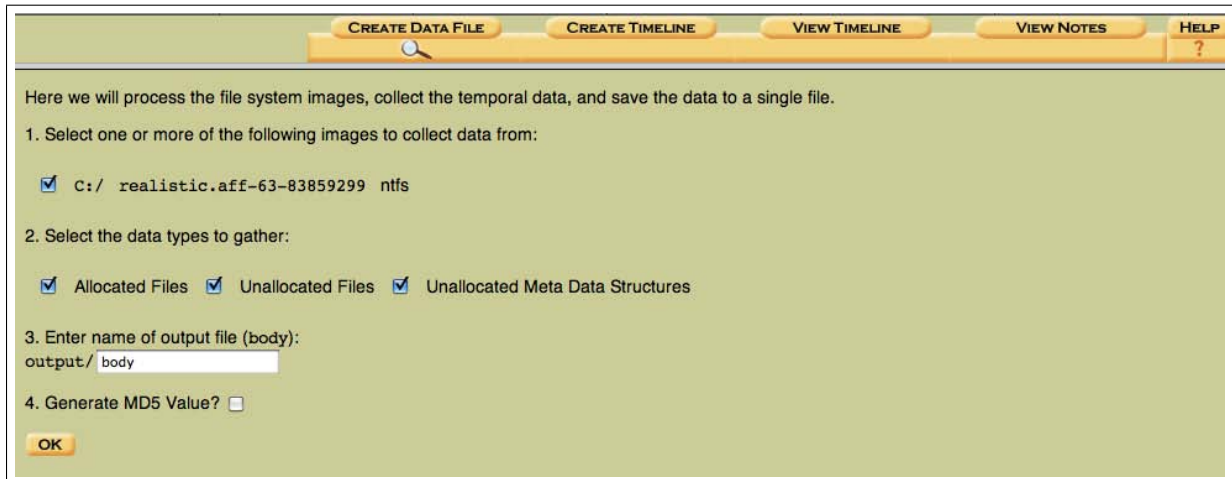


Figure 3.7: Autopsy user interface.

PyFlag had the worst results of all three tools, though it is more because of display and maturity of the software than capability. PyFlag allows the user to recover all the office documents if they explore the right directory areas, but it does not present these options in an easy-to-find manner. PyFlag did not extract any of the email or instant message communications by default. (The software is capable of extracting email and IM information from network captures, and this capability could be added to static media captures as well.)

None of the tools presented the user with information that there were multiple accounts on the media being imaged, nor did they attempt to sort the results by user account to provide identity context with results. This information is important in an investigation as it can provide attribution of specific data or plausible deniability if a user did not have ownership.

3.4 A Brief PyFlag User Study

We conducted pilot user study to test the usability of the PyFlag interface for the average user. Two computer science graduate students were selected to perform a very basic analysis of a 64MB USB flash drive with the files deleted. (A small flash drive was used so as to avoid overwhelming the students with the large realistic disk image.) The drive was formatted in FAT32 and the files deleted using the Windows command line `DEL` command. The drive was then imaged using `dd`. We also analyzed the drive image to verify each file could be recovered.

The primary goal of this pilot was to analyze the current PyFlag interface from the point of view of a non-expert user. The non-expert user is often overlooked by forensic tool developers,

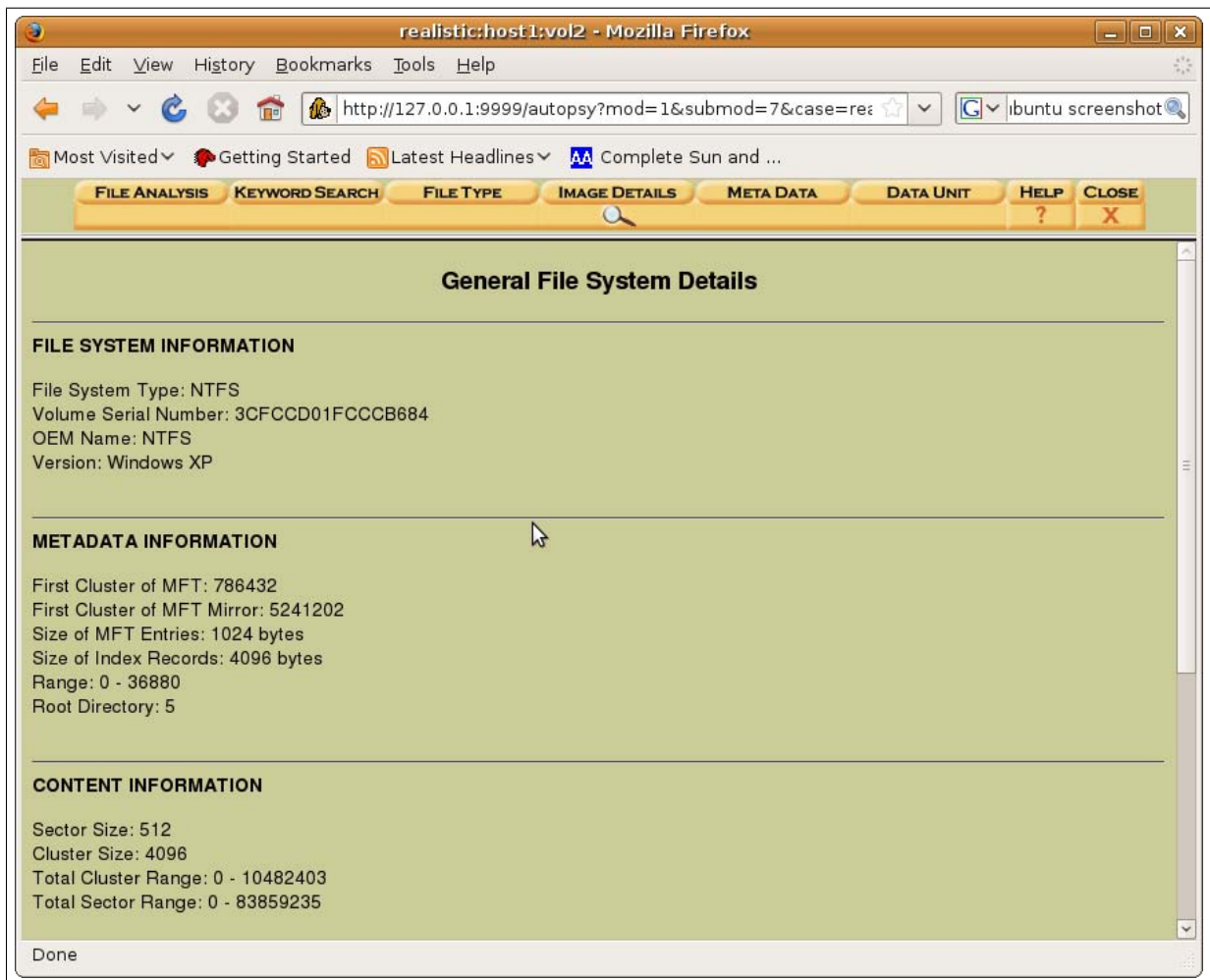


Figure 3.8: Autopsy general report.

yet this is important. A case investigator often knows what information they are looking for as evidence. It could be a document containing a phone number, credit card numbers, or email communication between certain people. It could also be certain kinds of images, such as child pornography. In these cases, the case investigator may not need a digital forensic expert to find the files, they need a computer user to operate the tools available. By designing the tools in such a way that provide a forensically sound manner yet still retain ease of use for “standard” investigating, the workload could be distributed across a larger group of users. Only in cases where the subject has used methods to hide data or in cases where advanced techniques are necessary, would a forensic expert be required to step in.

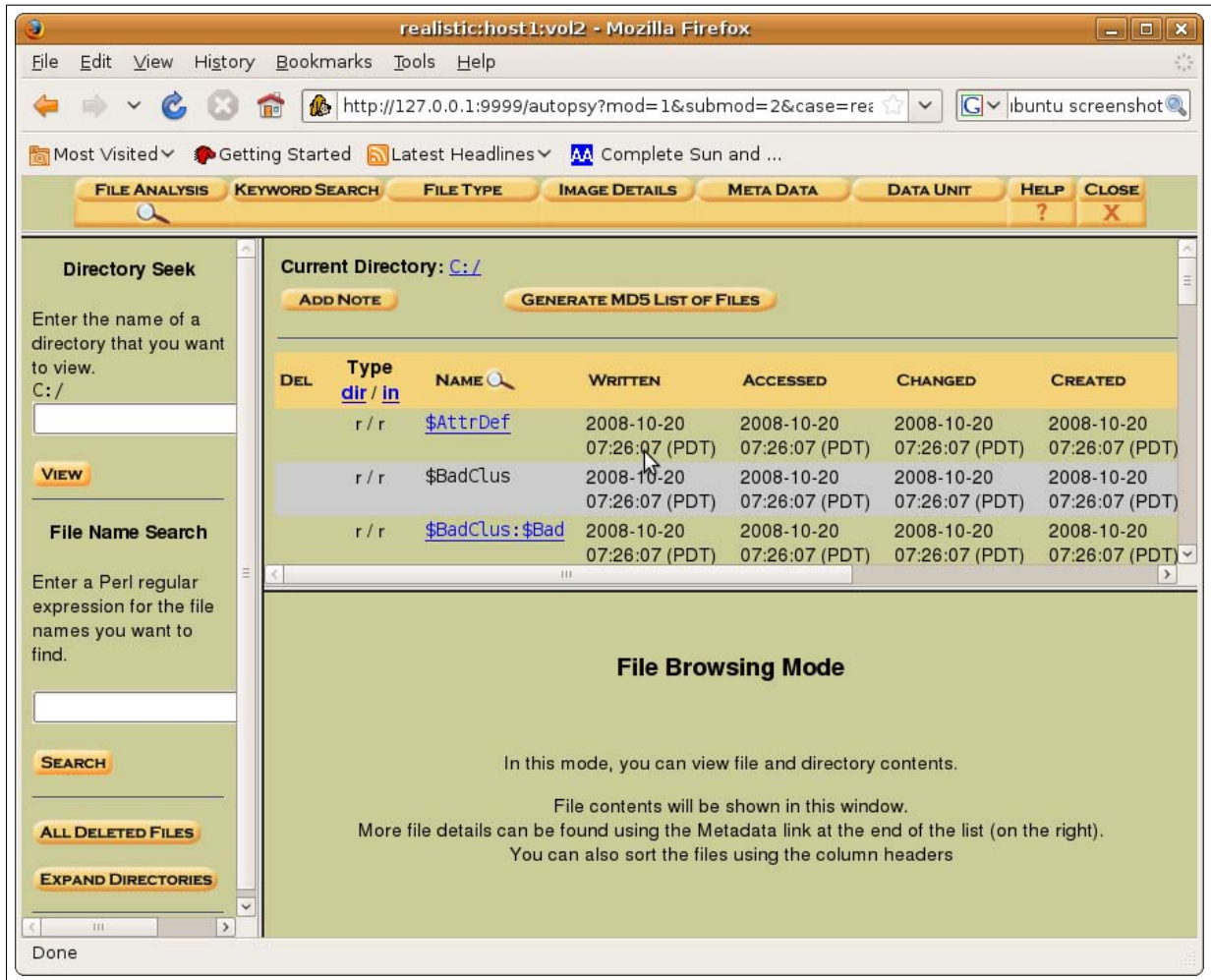


Figure 3.9: Autopsy file browsing.

The second purpose of this test was to analyze the current PyFlag reporting format from the point of view of a non-expert user. The current reporting format uses tables to show results. Each table is presented as a frame in the browser window, with the PyFlag navigation bar at the top. The table may also be alone in a separate browser window. The table view is powerful in that it lets users filter by every attribute PyFlag has recorded through a point and click menu, and view the raw SQL query that generated the results, but it is designed for presenting tabular data in a fixed format, not for data fusion and readable display.

The list of tasks performed by the users appears in Table 3.3; The steps required are listed in Appendix with screenshots. User results and responses were then recorded quantitatively via Likert Scale scoring and qualitatively via conversation style dialogue where the author asked the

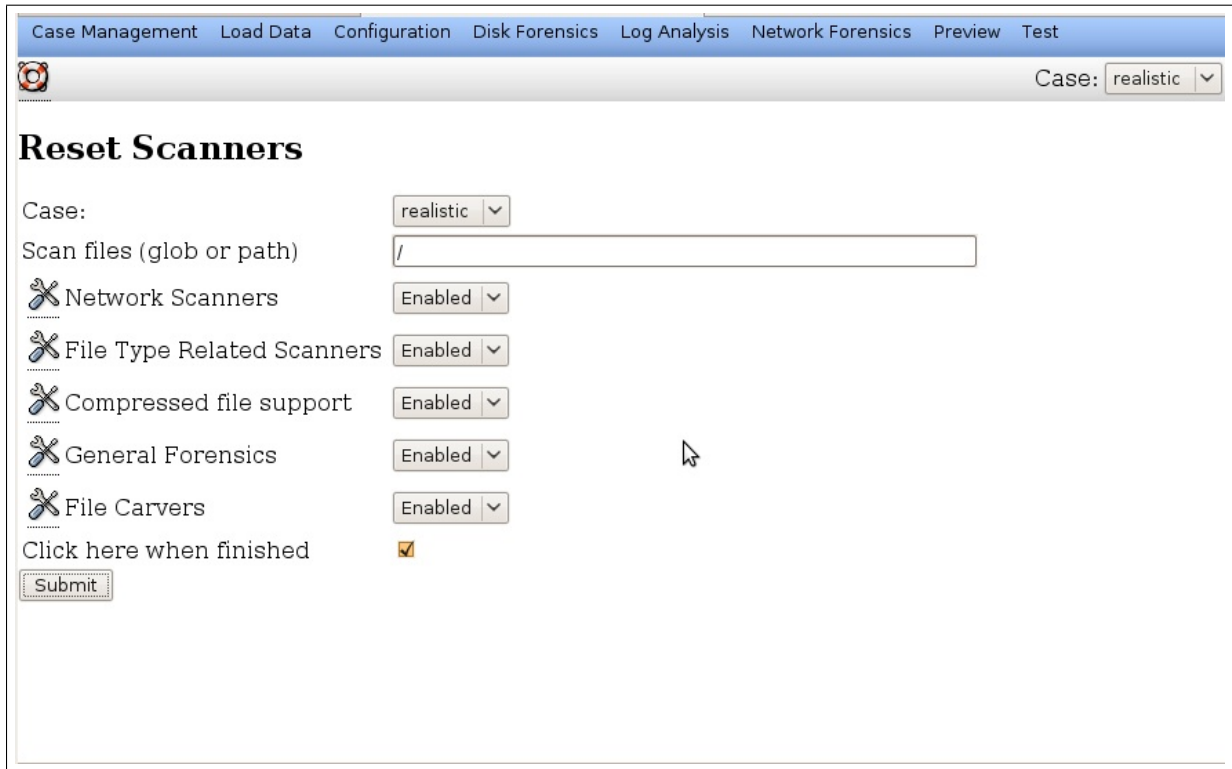


Figure 3.10: PyFlag Scanning Options

user's opinions, what they liked and disliked, and what they would change about the interface for a better experience.

At the conclusion of the tasks the students were debriefed. Based on their comments and difficulties accomplishing the tasks with the provided software, we drew the following conclusions:

- PyFlag lacks a summary report to give a snapshot of the image being examined. Such a summary report would be useful.
- The generic report option allows the user to select from every attribute capture by PyFlag in the back end database. This allows useful information to be reported on along with the generic information per image, but overwhelmed the users with too much information.
- Filtering is applied in a separate step once the search is conducted. Though the user is able to view the raw SQL query, there is no option to save the query and filter for use on future cases. The option also assumes a familiarity with SQL; the students were both

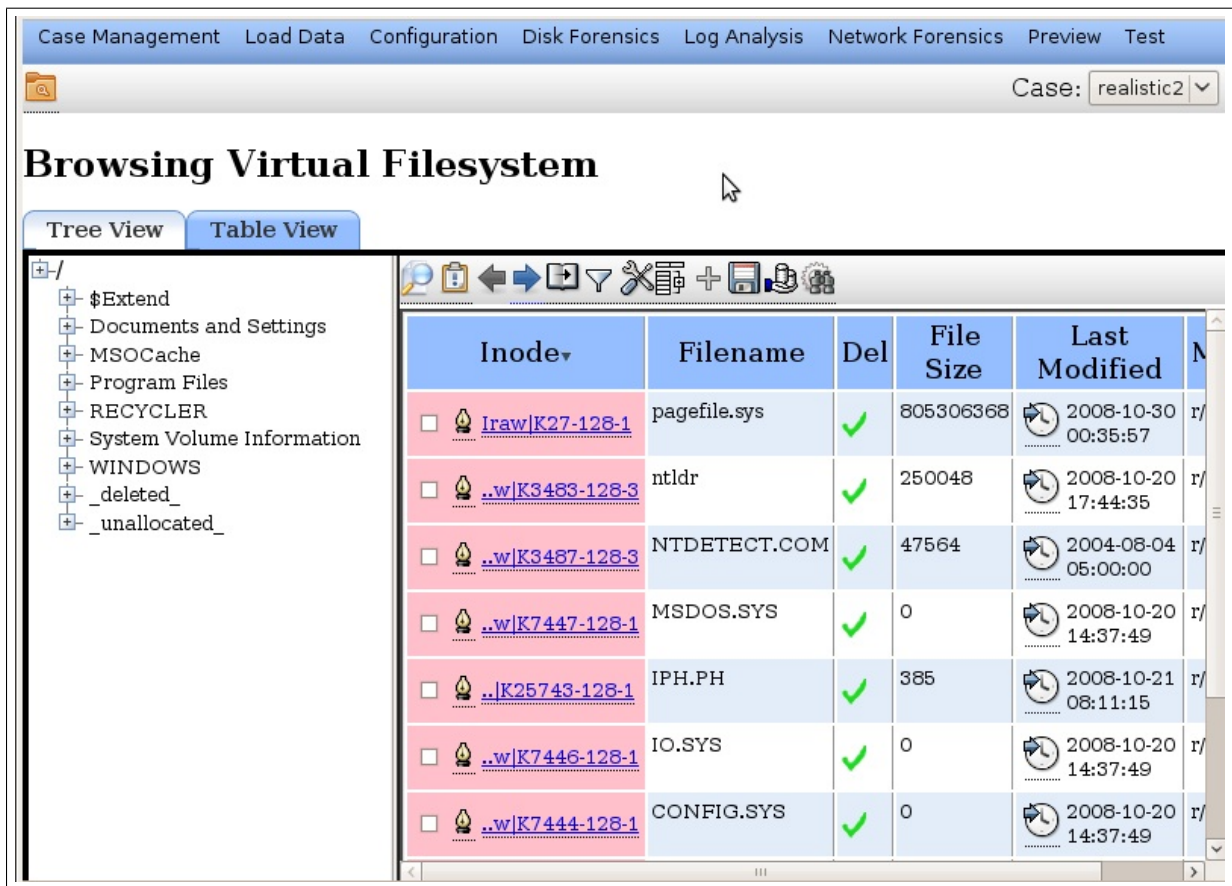


Figure 3.11: PyFlag File View

intimidated and surprised by the SQL and were not able to make use of it in the way that the PyFlag authors intended.

- For PyFlag, the Disk Forensics menu offers an option called “I’m Feeling Lucky,” (similar to the Google search option). This menu option allows searching for certain communication streams, html pages or image files in the ingested image. Calling this option “I’m Feeling Lucky” caused the users to skip over it; this was unfortunate, because it contained the most useful reports for the standard tasks at hand.




Case Management Load Data Configuration Disk Forensics Log Analysis Network Forensics Preview Test				
Case: realistic2				
Thumbnail	Filename	Type	Size	Times
 Broken <input type="checkbox"/> ..9879424:36864	/_unallocated_/o00000517	Targa image data - RGB - RLE 40 x 71	36864	00:00:00
 Broken <input type="checkbox"/> ..3209216:40960	/_unallocated_/o00004480	PCX ver. 2.5 image data	40960	00:00:00
 Broken	/_unallocated_/o00005160	Bio-Rad .PIC Image File 25938 x 18023, 27753 images in file	49152	00:00:00

Figure 3.12: PyFlag Image Report

- Create a new case.
- Ingest a data image
- Load a file system image
- Scan the image
- Produce a report of the file timeline of the data image
- Produce a report of the deleted items of the data image
- Produce a report of the images on the data image

Table 3.3: Tasks performed by the users in the pilot PyFlag user study

CHAPTER 4:

A Vision for Automated Media Reporting

As shown in Chapter 3.2, the current state of forensic tools are designed with the expert forensic user in mind with little desire for cross case correlation or identity resolution. Time requirements and limited expert man hours are not considered when these tools were designed. Most tools involve a considerable learning curve for which companies sell specialized training.

Better tools need to be developed to provide fast triage of large data sets. Automated ingesting of images, searching for known files or files of interest, and standardized reporting on the images are all functional requirements of an ideal system and all quite easy to automate, given the desire, programming skill and budget.. These functions could provide an untrained investigator with answers to potential questions about data that resides on the media, yielding information relevant to the case.

4.1 Automated Ingestion

Ideally, an automated system will ingest a media image with no or minimal user interaction[3]. The image would then be scanned for a partition table and overall drive information. For each partition in the table, or if no partition table exists, a file system should then be tested for. Once file systems are identified, the partitions would be ingested. The system would have fail safe fallbacks, such as a ingesting media as a raw binary stream, assuming nothing about partitioning or file structure.

This level of automation is easily achievable. Sleuthkit's `mmls` command lists each partition on a disk image by type, offset and length; Sleuthkit's file system commands (e.g. `fls`, `istat` and `icat`) take these offsets as parameters from the command line. But the open source community has generally not been interested in automating the process. For example, PyFlag has menu options for printing the partition table and selecting which partition to ingest—but only one partition can be selected at a time. If the user wishes to process multiple partitions, multiple ingests must be performed on the same image.

As before, the goal is to automate trivial tasks that can be abstracted or generalized. A dedicated workstation or acquisition device should be able to automatically perform these steps on all devices connected, creating a sort of “drop center” where large amounts of collected devices

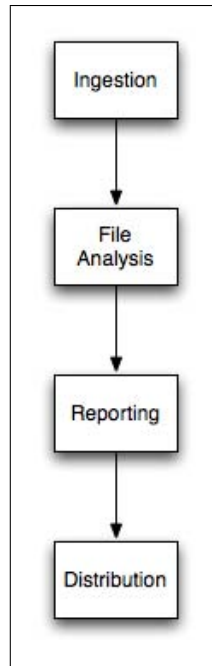


Figure 4.1: An Automated System State Diagram

can be connected and acquired without further user interaction. In the case of criminal investigations, it is unlikely that the same device will be acquired more than once due to evidence handling requirements. Automated systems should detect and warn when the same device or image is presented more than once for analysis.

4.2 Automated File Analysis

Once ingested, an image should have its resident and deleted files automatically analyzed. (PyFlag performs carving at this point as well, although this is not strictly necessary.) In this framework, the end user is a non-technical investigator looking for information about the users of the acquired devices. This analysis step should attempt to organize data to answer questions the investigator has, such as who owns this device, who are they communicating with, are they linked to other people being investigated, do they possess illegal or interesting files?

Automated analysis can be broken into several categories:

1. General searching and analysis should be done for popular file types unique to communication, internet and media activity.
2. Known file analysis should be conducted if there is a known corpus of evidence being searched for.

3. Deleted files should be discovered and recovered if possible, sometimes through carving.
4. Finally, malicious software should be searched for and identified.

General searching and file analysis can provide a high level overview of ingested media. Basic histograms can be constructed by file type using the built in extensions of the native file system of the image. Therefore, the initial file system should be walked, with file information being recorded in a searchable structure. By performing this task once, more in-depth follow on reports can be generated without additional processing of the image. A simple database query can be conducted in the reporting step.

Compressed and container files provide a particular challenge during this phase. The architecture must allow file introspection to determine the contents of these file types. The basic computer science principle of recursion must be applied, as container files may be recursive themselves.

File extension validation can also be performed during the analysis phase. There are several methods to verify file type used today. One approach is to base file identification on “magic numbers” or signatures appearing at the beginning or end of the file, a technique used by libmagic[10] and Scalpel[23]. This approach can be extended to use regular expressions, as PyFlag does. PyFlag runs file identifier searches on each file, with each known file type scoring the suspected file. If unknown file A is scanned by known file type scanners C and D, both C and D would return scores that A is of their known type. The file is then assigned the type of the highest score received. A mismatch between the file extension in the file system and the scored file type might indicate attempted data hiding, or a non-standard system that requires more in-depth analysis. (One significant problem with the PyFlag scoring approach is that the scores generated by each recognizer must be manually tuned by the developer.)

Deleted files should also be searched for and recovered if possible. Files that can be recovered by file system analysis should be recovered instantly. For all unknown blocks or inodes, carvers should be run to maximize the data recovered.

The analyzed file system and deleted files should then be scanned for known files or data of interest. In cases where there is sufficient knowledge of the crime committed, the investigator may be searching for presence of some data. This can range from malicious exploitation software, to child pornography or communications to certain email addresses. This scanning

can be conducted automatically by the system, and speed can be increased by conducting file hashing during the initial analysis and then comparing hashes. (We have previously shown that Bloom filters are an ideal data structure for this step for their fast access time and ability to have low false positive rates[16].)

Installed malware should be searched for by the system. The presence of malicious software could prove significant to a criminal case. Malware such as backdoors may directly influence the legal requirement of reasonable doubt. Also, in cases where the media image was obtained while in transit, the presence of certain malware could provide clues to where the image came from or what kind of environment an image was used in. Malware can be identified from MD5/SHA1 hash sets, or using standard anti-virus systems such as ClamAV[32].

Finally, custom information extractors can be run against the image. If an operating system is detected, user accounts, user names, and other OS unique information should be extracted. The detection of communication programs in the previous steps should trigger extractors to provide account names, email addresses, communication threads and more. Image files should be analyzed for possible EXIF data.

4.3 Automated Reporting

The system must then automatically provide a useful report to the user, converting acquired data into useful intelligence for the investigator. The generic report should have several sections, to include user accounts, communications, internet history, unique images and video, and a general typing of software installed. The report itself should be modular, so that parts may be added and removed as the system develops and technologies change. Following is a sample of currently useful reporting sections.

The **user accounts section** should be a small scale identity resolution report on the current media image and the corpus of previous analyzed images. If possible, account names should be mapped to real names, email addresses, and online aliases. Ideally, the report should let the investigator know if the drive was used by one or many persons.

The **communications section** should be built upon the user account section. Each communication account owned by a user account can then be queried for additional information. Email histories can provide a wealth of information regarding most frequent email subjects and persons communicated with. Histograms for most frequent results and most recent results are likely to provide useful information to the investigator.

For each communication program that the system is able to identify, a graphical representation of the captured snapshot of communications could also be useful to the investigator. A mailbox file, or saved instant message communications provides a wealth of knowledge that can be extracted. From this data, a network, or mesh, diagram can be constructed with weighted edges to represent frequency of communication.

Internet history should be presented to the investigator but not overwhelm. As with email, the most recent and most popular sites should be displayed along with time stamps to establish a general context for the information. Cookie files should be parsed for cached account information and listed along with relevant extracted data from the file. As each site can implement the cookie file uniquely, this will present a problem to extraction development. If the directory structure is bare and the web browser has an access time indicating recent use, the report should make a note that private browsing or history erasing may have been conducted.

Media reporting (still image, audio and video) has seen significant interest by law enforcement, especially in the prosecution of child pornography. Media should be presented as most recent greater than a certain tunable size and largest by size. By putting a lower bound on media size, common thumbnail and embedded graphics can be ignored. Also, if applications are installed that are known to allow file sharing, their downloaded contents should be presented to the investigator. Media should be presented in thumbnail form to allow fast analysis by the investigator.

Media reporting can benefit tremendously from an aggregated corpus of other media. As the system ingests more data, it should keep track of how often file hashes have been seen and where. The system can automatically generate relationships and perform identity resolution between ingested media, providing the investigator additional vectors for analysis. Common files can be relegated to background noise to further streamline and tune reporting.

As mentioned previously, the report should provide a general summary of software present on media ingested. Software should be listed under general categories such as Peer to Peer, Communications, Encryption or Software Development. By listing known programs and providing a context of what is installed on a drive, the investigator is given information to form hypothesis for the case, rather than raw data.

A mock up report is presented in Appendix A. Sections explained above are present, providing an instant report on a sample drive allowing the investigator to perform their analysis in minutes.

The report should be as human readable as possible. Technical information pertinent to how the data was found on the image should be hidden in the report, but stored in the framework database to aid further in-depth analysis if warranted and provide the legal requirements necessary to recreate the finding in a courtroom.

4.4 Report Distribution

Finally, the generated report should be distributed. Here it must be decided who is to receive the report and how it will be delivered. The investigator assigned to the case will normally be the recipient, though the presence of certain data may require other recipients. Based on the known file searches conducted earlier, other parties may receive a result, such as FBI involvement with the presence of child pornography, or an expert digital investigator with the presence of encryption software or malicious utilities. The report can be delivered in a variety of means, though to preserve data integrity the analysis system will normally be air-gapped from real-world networks, making electronic distribution difficult. Printed copies are the most likely form, with cover pages addressed to the required recipients. Though this step requires human intervention, it can be performed efficiently by untrained personnel with minimal cost or time requirements.

CHAPTER 5:

PyFlag Implementation

To create the framework for automated ingestion, analysis and reporting, this research project used PyFlag as an underlying engine. We did this despite the apparent usability problems with PyFlag because PyFlag provided much of the mechanisms required for an automated reporting system—file system handling tools, file identification, extractors, carvers, an underlying database, and a job scheduling system.

The disadvantage of using PyFlag is that significant time had to be spent learning how to use PyFlag and the results of this work are not directly usable by other organizations that do not have a working PyFlag installation. We have attempted to mitigate this by distributing a VMWare virtual machine with PyFlag pre-installed. However, this is just a work-around, and ultimately, to be useful, either the usability problems with PyFlag must be address or the reporting system will need to be separated from PyFlag.

Another option that we considered was to structure the automated report as a series of EScripts for Encase. Encase provides a scripting language that supports customized searching within the Encase program. Programmers are able to describe their own file classes and how to extract information from the files into the class. Encase supports file identification through the file extensions, but we were unable to find examples of file categorization based on file header information or content based search. Encase scripting allows the user to apply scripts to all files within sets of file extensions, files of certain sizes, other file attribute conditions, or all files currently under examination in a case. To apply multiple custom extractors to all files of a certain type regardless of extension requires examining all files in the case for each extractor, which increases linearly with each custom extractor, a solution that is time consuming.

Also, Encase is a commercial product with a high price tag, closed source, and has significant hardware requirements. Even though we have Encase in our lab, the cost of a single Encase license discouraged us from choosing it for development for automated reporting: the cost can hinder significant deployment to smaller environments. The closed source code of Encase was also a limiting factor. Though Encase provides a scripting language which can be run from inside the application, we were unable to find a method to automatically ingest, analyze and report with a minimum of human interaction.

To learn more about developing for the PyFlag system, we created an AOL Instant Messenger Log identifier and extractor, (PyFlag did not previously include support for this file format.) We also created a report that combined the information from our extractor with other information already provided by the PyFlag infrastructure to demonstrate a proof of concept for automated reporting with data fusion.

The remainder of this chapter discusses these two additions that we made to PyFlag.

5.1 AIM Plugin

By default, instant messaging programs do not keep logs of conversations when running. Despite this fact, these programs do have the option to turn on logging of all conversations and chat rooms. The presence of these logs can provide a detailed view of the user's social network. Because of this, we decided that creating a plugin for these formats would contribute to PyFlag. There were several steps to implement the AOL Instant Messenger file identifier and extractor. The first was to identify what the AOL logs would look like. Three popular AOL IM clients were chosen, the official AOL IM client for windows, Pidgin for Linux and Adium for Macintosh OS X. Each client was run with logging turned on, and several conversations were then created. Each client saved their logs in a different format, HTML for AOL, text for Pidgin, and XML for Adium. The logs were analyzed for both literal and regular expressions to match. The log files for each client are available in Appendices C, D, and E.

PyFlag performs file recognition and extraction as two separate steps. First, files are recognized via the Magic class, named from the unix style `libmagic`. The Magic class implements recognition through literal and regular expression matches. PyFlag extracts information through the `Scanner.GenScanFactory` class. This scanner implementation states what kinds of files it should be run on, and then when run, it performs the actual extraction and storage of data.

5.1.1 PyFlag's File Identification Hierarchy

PyFlag implements a main registry of file handlers called `Magic.MagicResolver`. When PyFlag initially scans a disk image, it creates a virtual tree of the actual file structure as well as deleted files and unallocated inodes. PyFlag then runs its file handlers against every node in the tree to attempt to identify the file type. These identifiers inherit from the `Magic.Magic` class. Each identifier must provide a regular expressions or literal rule for the file type it should identify, and then provide several samples of matches for type validation. Each node in the tree receives a

Field	Type	Null	Key	Default	Extra
inode_id	int(11)	NO	PRI	NULL	
packet_id	int(11)	NO		NULL	
session_id	int(11)	NO	PRI	NULL	
transaction_id	int(11)	YES		NULL	
nick	varchar(50)	NO	PRI	NULL	
user_data_type	enum('target_msn_passport','user_msn_passport', 'display_name','url_enc_display_name','locale','os','client', 'contact_list_groups','home_phone','work_phone','mobile_phone', 'msn_mobile_auth','msn_mobile_device','forward_list','allow_list', 'block_list','reverse_list','pending_list','added_user_to_list', 'login_time','lang_pref','email_pref','country_code','post_code', 'gender','kid','age','birthday','client_ip','personal_message')	NO	PRI	NULL	
user_data	text	NO		NULL	

Table 5.1: PyFlag msn.users table in MySQL

Field	Type	Null	Key	Default	Extra
inode_id	int(11)	NO		NULL	
packet_id	int(11)	NO		NULL	
session_id	bigint(20)	YES		NULL	
sender	varchar(250)	YES		NULL	
recipient	varchar(250)	YES		NULL	
type	varchar(50)	YES		NULL	
data	text	YES		NULL	
p2p_file	int(11)	YES		NULL	
transaction_id	int(11)	YES		NULL	

Table 5.2: PyFlag msn.session table in MySQL

score from each Magic handler, and the highest overall score labels the file as the corresponding file type in the corresponding cases SQL file table.

We created three new classes, one for each log file. Each class inherits from Magic.Magic. Regular expressions were developed for each log format as well as literal matches that occur in each log. The literal matches were given a higher score than the regular expression matches as they occur less frequently and are unique to the application logs. The Magic handlers then label each file as Instant Messenger Log Files. A possible side effect to this strategy is that any communication or file that discusses these log files may also be matched by the Magic handler.

5.1.2 PyFlag's Extraction Hierarchy

Data Extraction in PyFlag is handled by the `Scanner.GenScanFactory` class. For each log type identified above a new class was created that inherits from `Scanner.GenScanFactory`. These classes specify what file types to be run against, in this case Instant Messenger Log Files. These scanners read the files line by line, and perform regular expression matching for extraction. This step is similar to that performed in the previous `Magic` class, but only for these basic text files where the same regular expressions that hold data are the same that identify the file type. These scanner classes then make a database connection and insert the data into the proper tables for later analysis.

One issue discovered is that the AOL log format does not put the username that is logging in the log unless they actually respond. For example, if **user2** sends **user1** several messages but **user1** does not reply, the log only has the IM name of **user2**. This can be resolved by pulling the **user1** name from the path in the filesystem where the log exists, but if the log file has been deleted and is recovered via carving, there will be no way to recover **user1**'s IM name.

Once the log files were identified and the user names extracted, they need to be placed into the PyFlag database. The current schema consists of a table for MSN users and MSN sessions. The PyFlag maintainers recommended placing all instant message style communications into the MSN table, despite the varied protocols in use with different fields. The MSN tables in PyFlag are designed for network based stream captures instead of static log files captured on media. The `msn\ users` table stores the nickname of the sending user, but not the nickname of the user a message is sent to. The `msn\ sessions` table is designed to store the TCP/IP information about the MSN communication in the sender and recipient columns, rather than the usernames involved.

5.2 Report Plugin

We then implemented our idealized report. The current PyFlag database provides much of the data required to generate our report and we leverage from it heavily. The PyFlag reporting plugin architecture is unable to display a report in the desired format. Our findings and discussion of how we constructed our code follows.

5.2.1 PyFlag's Reporting System

PyFlag's reporting is the most primitive part of PyFlag. The reporting system consists of small scripts that perform SQL `SELECTs` on the database server and present the results in HTML tables in a web browser. The system even has provisions for presenting the SQL to the user so that it can be modified as necessary to improve the specificity of the query. PyFlag also has the ability to interactively build an SQL query through the use of its "manual" report.

PyFlag offers no ability to perform automated correlation between cases. As with the other tools examined, users are allowed to add multiple disk images, or sources, to an individual case within the system. There are currently no PyFlag utilities that provide cross case indexing, searching, or notification that unique files or email addresses have been seen previously. Though information protection and hiding is important to provide forensic soundness and follow legal guidelines, we believe this can be implemented and still prevent contamination between cases.

PyFlag also offers no ability to perform *identity resolution*. A relatively simple approach would be to have the system group together all of the email addresses that have the same username but at different mail services. For example, **user1@hotmail.com** could be grouped automatically with **user1@gmail.com**. Likewise, there is no provision for telling the system that two email addresses represent the same person.

5.2.2 PyFlag's Databases

PyFlag implements one global database table called `pyflag` upon creation. Inside this database are tables for geographic IP address lookups, `WHOIS` information for DNS resolving, a dictionary used to store keywords for indexing cases, and other information used to provide information to cases. This database is populated during the installation and first run of PyFlag, and the only table modified by any other case is the job table. This table keeps track of queued jobs to be run in other cases and is emptied when complete. PyFlag also creates a global database called `nsrldb` to hold the NSRL RDS information if it is loaded by the user.

5.2.3 The Reporting Program

We designed our report in python so that future integration with PyFlag would be simplified. As PyFlag currently limits reports to the table view shown previously, we decided to run our reporting program against the PyFlag database directly and output into html. This maintains the constant interface used with PyFlag and does not require the user to learn varied applications.

GUI_filter_history
LogicalIndexOffsets
LogicalIndexStats
MainThread_12327000201
MainThread_12327000261
MainThread_12327000281
MainThread_12327002721
annotate
block
connection
connection_details
dns
email
file
filesystems
ftp_commands
ftp_data_streams
ftp_sessions
hash
http
http_parameters
http_sundry
ie_history
inode
interesting_ips
iosources
irc_messages
irc_p2p
irc_session
irc_userdetails
log_tables
mac
meta
mmsessions
mozilla_form_history
mozilla_history
msn_p2p
msn_session
msn_users
passwords
pcap
reg
regi
reporting
resident
sql_cache
sql_cache_tables
timeline
type
virus
webmail_attachments
webmail_messages
xattr

Table 5.3: PyFlag tables in MySQL

Our code is organized into the logical sections of the idealized report. Each section queries the appropriate current PyFlag database tables and extracts the desired data. The organization of the data and display attempts to turn this data into information for the user (or investigator) so that by reading through the report once, they have a generalized understanding of the media being examined. Our reporting code makes a database connection to the PyFlag database, runs a custom query on the case specific database, organizes the data, and outputs into html. The user must then open the html report for viewing. This reporting code is not meant as a final product report ready for deployment, but is meant to provide proof of concept that the PyFlag architecture allows rapid third party development to meet custom requirements and can implement the automated reporting framework discussed in this thesis.

5.3 PyFlag Limitations

While working on this project, several limitations with the current design of PyFlag became apparent. As a new developer trying to work with established code, we found several difficulties with the existing implementation of source file organization and internal database schema. Also, while using the PyFlag suite to perform forensic analysis, we found user interface and design decisions that we believe deserve more attention.

5.3.1 Fragmented Database Schema

PyFlag's current design stores communications in a variety of different database tables. Some tables are used for storing the results of multiple extractors, while other tables were created for use by a single extractor. This haphazard design of tables appears to be the result of the system's organic growth and reflects the lack of an underlying system architecture.

For example, PyFlag stores information about email messages in a table called `EMAILS` but stores MSN chat logs in a table called `MSNMessenger`. The developers recommended that the AOL extractor store its logs in the `MSNMessenger` table as well, and craft its results to fit into the schema that the `MSNMessenger` system had established. This makes it difficult to write reports that evaluate all communications between a pair of data subjects, as the MSN tables are designed to store one way MSN messages as part of an overall stream. The table stores who the message was sent from, but not who it was sent to. The fragmented schema also complicates the maintenance and future development of the system.

Reflecting the design of other forensic analysis tools, PyFlag stores evidence in containers called *cases* with no provisions for linking information. Searching for an email address requires explicitly searching each email table for each active case—there is no single table of all email addresses that the system has encountered. The advantage of this approach is that the performance of the system in analyzing any specific cases does not degrade as more cases are added to the server; the disadvantage is that searching for a single email address in *all the cases* is a slow and (currently) manual process. This reflects the development of the system for managing evidence in police cases, rather than for performing intelligence investigations.

5.3.2 Reporting View

The PyFlag standard view for all information is via a table viewer. All data is retrieved from the PyFlag database via SQL commands and displayed in table format to the user. This view gives

power to the expert user as they can customize the SQL queries in the view to fine tune their information. This table based view seems derived from the developers deciding how to present the data stored to a user, instead of asking the users what they want to see and crafting the view around that desire. Currently too much data is displayed by the built in reports, and even after customizing queries, the display is still framed by the table and column results of the database. Presenting information in an easy to read format while still maintaining the underlying database structure must be addressed to allow the average investigator to benefit from the system.

5.3.3 Difficulty of Integrating New Tools

PyFlag lacks a standard interface for describing new tools—for example, a new file system implementation, a new carver, or a stenography detector. Instead, each tool must be custom developed within the PyFlag architecture with its own set of Python classes. It would be useful if there was a way to easily add additional tools by simply adding lines to a table or an XML file.

Once a new tool is introduced to a system and verified to work, it can then be added to tools that PyFlag is aware of. Each tool could be described to PyFlag with a name, path to execution, necessary flags, file types to be run against, maximum run time per file (to determine if the tool has frozen and should be abandoned), and how to parse the return data or file to be PyFlag aware. This tool description methodology could allow PyFlag to be highly flexible for new or custom written tools that programmers did not intend or were unable to implement in PyFlag directly.

CHAPTER 6:

Results on Realistic Data

We ran the proof of concept extraction plug in and reporting utility against the realistic data set previously created to simulate real world results. Ideally, we want to present the user with information from the image instead of large amounts of technical data with minimal user interaction between choosing the device to image and receiving the report.

The reporting utility generated the report in Appendix B. Though rough in presentation, the report presents the core ideas of automated digital forensic reporting and proves that PyFlag is an appropriate framework for future development.

The first section provides signature matches. In this version, the report is looking for matches against the NSRL RDS. However, custom hash libraries (and bloom filters for distributed remote applications) can store more relevant information to the individual investigator's concerns or their organizational interest. As most signature searches are for known data features or files of interest, the significance of a signature match requires that it be the first thing to appear to an investigator in the report.

The next section of the report, users, is broken up according to the main users of the system. The current report is focussed on a Windows XP system and automatically filters out the default users the operating system has installed. For each user on the realistic system, Administrator, domexuser1, and domexuser2, the examiner is provided with information about what the account is primarily used for, whether it be media related, software related, or other uses. Each user has reports on several subsections.

The first subsection is file related. PyFlag supports UID information to be extracted from the filesystem, but the current implementation is unreliable at best, often showing all users as UID of 0. Therefore this report relies upon the file path stored in the database to determine ownership. Once the user id, group id, and permissions are extracted by PyFlag, future reports could take these into account to determine likelihood of ownership of all files in the image. The report shows the most recently used files as well as the most popular file extensions in that user's account.

The following subsections cover other areas of interest to the investigator. Web history, Images, Document Files, Emails and Instant Message data all have subsections for each user. We were unable to extract image thumbnails from the PyFlag framework at this time. By organizing in this manner, and providing the most popular or most recent pieces of data, the data becomes information to the investigator.

CHAPTER 7:

A Proposed Framework for Automated Reporting

Based on the experience of working with PyFlag and other forensic tools, this chapter presents ideas and a design for a new automated reporting system.

The framework and implementation for automated forensic reporting was designed using a process based on user-centered design. By examining available tools, real world exploit reports and interviews with current users a better understanding of how forensic examiners are analyzing media was formed. From this data, a sample report was created, to show a simulated end product of analysis. The idea was to create a simple, easy to understand report that contained the specific information needed by an investigator. The report is also designed to provide vectors for further analysis, should more in depth data be required. From this report, a list of desired information was created, along with important properties and metadata.

7.1 Requirements

The requirements to achieve this automated reporting vision can be broken into three broad categories. First is system requirements for parallel processing, distribution, storage and functionality. Second is the user interface requirements to allow minimal interaction to yield maximum information in a reasonable amount of time. Lastly is the ultimate report requirements which converts the data acquired into intelligence for the investigator.

7.1.1 System Requirements

The forensic framework should provide several key functions. To provide a large corpus for identity resolution, the system should support multiple ingestion points for data. As PyFlag and EnCase already implement, parallel analysis must be performed. As PyFlag and FTK already implement, a database backend must be used to provide standardized data storage, data relationships and the ability for other programs to interact with the data. Finally, the forensic framework should allow a stand alone capability for field deployment.

Multiple Ingestion Points

The large amount of digital media retrieved in many cases makes data ingestion a significant problem for a forensic framework. At the ingestion stage, basic case information must be

entered for proper tracking and labeling of the case evidence. The media can then be ingested into the system for analysis. The data should ideally be stored on a high availability data server with fault tolerance with a high speed interface. The system can then run its analysis and extraction utilities against the centrally located data, reading into local memory as necessary. No analysis will be conducted on the ingestion systems during ingestion. Once the whole media image is received into the storage system, the framework will begin to create jobs against that data so that all operations can be controlled and tracked.

A single ingestion point creates a chokepoint or single point of failure for the initial step of this framework. At any given time, there can be multiple cases being prosecuted where media devices have been confiscated for analysis. The benefits gained by a large corpus of data require that all results be stored on a single system, though that system may consist of multiple computers networked together. Multiple systems that synchronize at specified intervals will result in a time window where data exists in one repository and not another. Reports generated on one system during that window will not be analyzed against data entered into the other systems during that windows, resulting in missed connections that may be found later if another report is run. Since a single system for an organization is the ideal implementation, that system must provide multiple ingestion points to the system to support an unknown number of users, large numbers of devices in a single case, or simple surges of new devices to be imaged.

Parallel Analysis

Imaged media is becoming larger and larger, creating more space to be searched and analyzed per ingestion. Regular expression searches must search through the entire address space of the device, while file carving searches all unallocated blocks and attempts to fit them together. The algorithms to perform these searches can be slow, and large devices can take days to analyze on average computer systems. Multiple media devices ingested simultaneously as required previously will only compound this problem.

Parallel analysis of the media when possible is the only way to implement a large scale forensic framework that can handle large amounts of media and provide results within a real world usable time frame. Serialized processing of multiple terabyte hard drives by a single computer would increase linearly with each new device added. The final report would not be produced until all drives were analyzed, a sum function on the time for analysis. Parallel analysis on independent systems would speed the analysis by the factor of number of systems available, but each system would yield a separate report, failing to perform identity resolution or any cross-

drive analysis. Parallel analysis of devices by a single system spread across multiple machines with the framework performing job management can obtain the speeds necessary for useful report timeframes with the benefits of a large system implementation.

Database Back End

To support the tracking of multiple ingestion points, job management for parallel analysis and storing the data from each case, a robust database backend is required by the framework. A relational database meets the previous requirements and has been demonstrated as the implementation of choice by PyFlag and FTK, though specific database selection varies. A database back end on a popular implementation also provides the ability for further tool development to run on data extracted into the database as well as run resolution algorithms to perform identity resolution on the entire corpus of information. A database also provides a large speed increase over a flat file implementation, while also providing atomic operations, journaled operation for recovery and clustering for maximum performance.

Stand Alone Operation

Analysis of media must be performed in the field In some cases. This can happen at checkpoints or random encounters where analysis of data in plain view could result in information relevant to law enforcement. The framework should support a stand alone method of operation, where all ingestion, analysis, and reporting is performed on a single device. Since time of analysis is a priority in this scenario, extensive analysis such as carving may not be able to be performed within a reasonable amount of time. Initial imaging for later in depth analysis should be a priority, followed by regular expression searches through the entire device. To provide lookup speed and secure distribution of data sets of interest, bloom filters should be recognized as a reference media to query for information of interest.

7.1.2 Report Requirements

The sample report in Appendix A shows all of the sample information to be gathered and presented to the investigator. This information should be arranged to answer questions the investigator has about the media device, listed in Section 4.2. The information of interest to answer these questions varies among user accounts on the local machine, email accounts, lines of communication, web history, and stored media. Each of the sections for answering these questions will be reviewed in depth below.

User Accounts

User accounts can reveal much in an investigation as well as provide for a means of attribution. Unix based systems provide a simple way to extract local accounts from a full system image. Unix, Linux and the BSD variants store user accounts in the `/etc/passwd` file, with hashes of the user passwords in the `/etc/shadow` file for linux or `/etc/master.passwd` for BSD. `/etc/passwd` also contains mappings from usernames to user identification numbers, or UIDs. An intact Unix based filesystem maps every file to the owning UID and permissions for read, write and execute for binaries. These permissions are stored in the format Owner(Read, Write, Execute), Group(Read, Write, Execute), World(Read, Write, Execute). Since the group permissions on a file could be significant on a multi-user system, the `/etc/group` file is also of significance to forensics.

While the `/etc/passwd` file may not map directly to real-world names, it can be a crucial link in this eventual mapping. Following are two examples where it could prove important, assuming a mailbox is write-able only by the owner. If a mailbox store is owned by UID X, that store can then be parsed for information. In typical mail storage systems, by running a histogram function on all email addresses seen, the address with the most occurrences tends to be the email of the mailbox owner, or the account the mailbox is for. Then, by parsing the individual emails, A regex for a greeting could be run against each email. If the greeting is of the form "Dear Name,," the regular expression could then match Name as the real world name of UID X. All files on the system owned by X can then be mapped to the real world Name. The same example works for saved html files, such as purchase receipts or forum account cookies. Using this approach, multiple aliases for email addresses, forum names, instant messenger names, and more can be mapped to real world names applicable to a case.

Windows based systems store the account information in either the SAM file or the Registry. If a FAT file system is used, no user account information is linked to individual files, while NTFS associates user accounts and permissions to individual files similar to Unix. FAT systems occur more typically with Windows 98 installations, while NTFS is the default for NT, XP, Windows 2000, Vista and Windows 7. For NTFS filesystems which store user information for individual files, the same mapping method used in the Unix based examples can be applied to Windows images.

In addition to extracting user information from system files and file system information, it is also possible to glean user account names from directory structures of home folders. This method is

easily fooled, as home directory names may be mapped to different user names, or have various one-to-many or many-to-one relationships to users.

Of note is that this method only works for complete operating system images. Removable media, such as USB thumb drives, CDs, DVDs, and most external hard drives may not retain the same UID mapping between systems, and reasonable doubt may be introduced into a case. For example, Unix and Mac OS X systems allow reading and writing of NTFS file systems via user-space driver. Furthermore, most removable storage is formatted with the FAT32 filesystem which does not support owner and group properties.

Signature Matches

Signature matching allows the investigator to be quickly alerted of files of interest. By creating a hash dictionary of alert files, such as child pornography or known hacker tools, the investigator is able to tell the system what files they deem important. By storing the hashes only, the system is able to save space and avoid having to keep copies of potentially illegal materials. Current hash implementations allow simple modifications to evade the filters[16], but may still provide useful results.

Beyond standard file hashes which most forensic tools already support, attribute hashing has potential for further development. By specifying a hash algorithm and storage mechanism, any searchable quantity can be distributed to investigators and searched for presence in a media image. For example, a collection of known criminal email addresses could be created, with each address then hashed. The set of hashes can then be distributed to the locations where forensic analysis is done for inclusion in the framework. The framework can then include a regular expression search for email addresses `[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}`. Each match can then be hashed and compared against the set of interest. The media can then be brought to the attention of the investigator for further analysis. Distributing the set of hashes provides an added measure of security where an investigator does not have the "need to know" all criminal email addresses, but is able to query against them and request further data.

Communication

Communications on media devices can reveal much to the investigator and can provide insight for further analysis. On digital media, the primary recorded communications are email and instant messaging. This data must be found automatically and parsed for information for the investigator. Trend analysis may occur, but primary account identities must be found.

Email may be stored on any media device, and with the advent of thumb drive versions of popular email clients, all media must be scanned for email. The representation of email on media is highly dependent upon the client application used. Microsoft Outlook stores email locally in a proprietary format called a personal store, or PST file. Apple Mail uses a proprietary storage format based on Maildir called Mbox.

Instant messaging (IM) provides tremendous insight into case investigation. Popular IM protocols are MSN, AOL IM, and Google Chat. MSN and AOL offer specialized client programs that talk on their protocols and implement all available functionality. By default, the MSN and AOL clients do not log their communications to the hard drive. Google chat uses the jabber protocol and was implemented as a Javascript application within the Google mail webpage, and stores all communication on the Google servers. For each of these protocols, several other client applications have become popular. Jabber, Pidgin, and Adium are all well used third party clients which implement subsets of the total functionality. All of these programs allow for conversation saving and automated logging in a variety of file formats, including plain text, html and xml.

Web Activity

Modern web browsers leave significant histories on media devices to provide a tailored experience for users. Internet Explorer, Firefox, Chrome and Safari all leave history files showing recently visited web sites. These browsers also allow the user to store a bookmarks file for frequently visited sites. These files allow a web profile to be constructed for each user, with special interest placed on web based email and forum sites where users can communicate.

In addition to history and bookmark files, web browsers also cache cookies for website authentication information and user tracking. Embedded flash media on sites also caches cookies, that most users are unaware of. Cookies contain web addresses and time stamps which are useful to establish a timeline in an investigation.

Privacy programs have become popular to erase a person's online tracks on their system. Safari and Chrome both provide a privacy browsing mode where no cookies or history file are saved to disk. Firefox provides a quick key combination to erase all private information. Based on this new functionality, web browser carving needs continued research to recover this information from hard drives and system memory.

Media

Digital media can provide a wealth of revealing information when examining a case. In this context, digital media is defined as digital photos, video, and audio. Digital photos have been of significant interest recently, though video is becoming increasingly prevalent with the increased number of phone/pda/camera/video recorder products being released.

Digital photos have the potential to store a large amount of metadata. The Exchangeable image file format (EXIF) standard developed by the Japan Electronic Industries Development Association (JEIDA) provides for metadata fields in images. These fields are normally used to record keyword tagging information, copyright information in professional images, camera settings for the photograph exposure, and even the serial number of the device creating the image.

Though recent real world analysis reveals that most consumer digital cameras do not stamp serial numbers into EXIF, many professional and high-end consumer cameras do. Also significant is the recent addition of Global Positioning Satellite (GPS) data to images directly from GPS enable devices as higher end cameras and camera-phones. The ability to apply this GPS data to a timeline view with map overlay could be beneficial to the investigator and is able to be completely automated.

By identifying common digital media files, the number of unique images on a file system can be identified to provide a kind of forensic image fingerprint. Common media files can help to identify an operating system, applications installed, or common websites visited. They can also prove membership in certain network subsets, such as mailing-list members that have shared an image, or part of a social network where a ‘viral video’ has propagated.

Attached devices provide another avenue of providing identifying attributes to a system. Operating systems store information about devices that have been attached, a popular case being Apple’s iPod media device. Windows operating systems store an iPod’s serial number in the system registry, allowing simple extraction. Macintosh operating systems also store a history of attached devices inside of the iTunes application.

7.2 How We Would Implement It

As shown before, PyFlag already implements many of the requirements necessary for the proposed framework. By choosing an industry standard database implementation, we allow for

fid	feature text	feature type (enum)
1	user@company.com	“email”
2	user@free.net	“email”
3	978-555-1212	“phone”
4	Jan 20, 2008 0600 EST	“time”

Table 7.1: Features Table

sfid	fid	reftime	case	inode_id	offset
1	4	Dec 31, 2007 1532 EST	case_1	P3 m3	1024
2	1	Nov 24, 2007 1200 EST	case_2	P10 m1	640
3	2	Nov 24, 2007 1200 EST	case_2	P10 m1	684

Table 7.2: Seen Features Table

row	identity	fid	decider	surity	reason
1	1	2	“email agent”	1.0	“vcard”
2	1	3	“email agent”	1.0	“vcard”
3	2	2	“email agent”	0.5	“account setting”

Table 7.3: ID Table

easier development, wider deployment and ease of upgrade. However, the database schema will be significantly expanded. We would also add a tool plug in option to provide rapid integration of new forensic tools and techniques. Finally there are other modifications to PyFlag itself to support the multiple ingestion points previously discussed.

7.2.1 Database Modifications

To implement our idealized model, we would retain the database repository of all case information. The idealized implementation will start with a modified PyFlag database schema. We will add a new database called “Identities” to the installation, aimed at feature correlation and identity resolution. This database will have several tables, namely features, seen features and ids. See Tables 7.1 through 7.3.

In this example, each feature that the extractors find in a case are placed into a separate database and table “identities.features”. Here the features get a unique feature id number (fid), the text of the feature and the type of feature. To speed searching, feature type should be a numeric and a separate table should map the number to an enum description, but it is presented in this way for clarity. When a feature is found by an extractor, the features table is searched for a previous occurrence. If the feature is new, it is given a new feature identification and placed into the feature table. The occurrence of the feature is then entered in the seen features table,

with the appropriate feature id, time and location information for this occurrence of the feature. For instance, if the same email address is seen in two separate cases, there will be one feature table entry with a feature id, but two entries in the seen features table referencing that feature id. This allows features to be searched through separate cases and may provide the examiner with useful leads for further investigation.

In addition to individual feature correlation, that id table database allows the architecture to manage identity resolution. In this example, two separate features, an email address and a phone number, are present in the ingested media within the same vcard file. A feature/identity correlation process would parse through the seen features table, examine the inode where the feature occurred for data types that it is aware of. In this case, a vcard file is a reliable indicator that an email would map to a phone number, so the correlation process would create a new identity number in the identity table and then reference these two features as separate rows to the same identity with a high degree of surity. The same correlation process may crawl through the features table looking for similar email addresses or name spellings and relate them in the identity table with a lower degree of surity.

Through this methodical mapping of features to identities, large sets of ingested data from similar cases could reveal patterns that would not be detected through normal manual forensic investigation. A unique date that is present in several different cases may indicate a relation though no direct path of communication has been discovered. Also, aliases within larger organizations could be mapped similar to nicknames, and with enough data, possibly to a real world name, one that would be present on a driver's license or passport.

To support this new schema and identity resolution process, we also propose a new PyFlag class for resolution, Feature Extractor. Similar to the Scanner class, the Feature Extractor would run on each recognized file in the ingested media. Feature Extractor subclasses would register with the main Feature Extractor. For file types that are registered, the main class would call the subclass on the file, with extracted features placed into the proper database tables as described above. This provides another level of complexity when developing new file identification and extraction capabilities for PyFlag.

7.2.2 Tool Plug-In

In addition to the new schema, we propose a plug-in architecture for other forensic utilities. These utilities would be obtained and compiled independently from the PyFlag installation.

Each tool would be described by a PyFlag configuration file to pass relevant information into the PyFlag generic tool extractor class that would wrapper each program. The description of each tool would be path to execution, execution flags, file types supported(i.e. for images a list of jpg, jpeg, bmp; similar to what PyFlag uses to generate it's automatic report of all images found). and then a method to parse the information returned. The resulting information would have to be specified if it is returned on the command line used to execute or in a separate file. This returned information can then be parsed with a regular expression and mapped to the PyFlag relevant fields for inclusion in the database.

For example, a steganography detection and extraction program that returns messages on the command line would be defined as follows:

Path to execution	/usr/bin/stegdetect
Execution flags	\$filename
File types	jpg, jpeg, bmp, gif, png
Return path	commandline
Parser	"Found Messages: " ([A-Za-z0-9._\%+-^]+) "\n"
Mapper	<table> objects
Mapper	<column> <communication> \$filename
Mapper	<column>

This approach has multiple benefits. First, as new applications are developed, either in the forensics field or by academic researchers, they can be integrated into existing PyFlag installations. Also, it allows third parties to write their own custom tools based on their organizational goals, but still use the PyFlag framework for overall extraction, data fusion and reporting.

7.2.3 Rich Reporting Output

Finally, we propose using a reporting output that is based on how investigators want to see reports rather than the table based output of the SQL database the data is stored in. PyFlag currently reports on single areas on a time, making fusion of different data types difficult. Summary reports should be automatically generated and presented to the user once media analysis is completed.

The additions presented in this section would bring much to the PyFlag architecture. Primarily, the PyFlag database would be organized for data correlation in addition to extraction, allowing large investigations and organizations with significant captured media to generate better infor-

mation from data analysis and provide more useful reports to the investigators. The framework would allow for rapid tool integration to adapt to changing technology without having to modify an existing in-place PyFlag installation. Finally, the reporting output would be focused on the end users and what they need from the framework, rather than how the framework is currently architected. Though not trivial to implement, we believe these changes are important and could be extremely productive.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 8:

Conclusions and Future Work

8.1 Conclusions

Currently available forensic tools are designed with the forensic expert in mind. The tools present a wealth of options and require specific and technical knowledge to extract data media images. These tools then present the extracted data in poorly organized fashions that try to show the user as much data as possible rather than prioritizing information according to relevance.

This thesis has focused on providing a snapshot of the system or media imaged as the most relevant information to a non-technical investigator. Basic functions performed on common operating systems with common applications are the focus, with in-depth analysis by trained forensic investigators still necessary but only in a small percentage of cases. As a result, this thesis has focused on common functions, communications, web history and media; such information can then be the starting point for additional analysis.

After reviewing current tools ability to ingest, analyze and report on a sample system image, we looked at the ability to modify the tools to produce an automated report. Though EnCase, PyFlag and TSK provide scripting abilities, each tool had drawbacks. While the scripting abilities with EnCase allow additional data to be extracted from images and added to reports, the scripting language is limited and there is no way to automate start to finish analysis and reporting currently. TSK also allows scripting using standard unix tools, but does not provide higher level analysis and extraction ability that PyFlag does provide. PyFlag however has poor database organization and report display, though of these three tools, it is the most advanced and easiest to modify.

Our proof of concept code demonstrates the ability for PyFlag to provide the automated reporting we believe will be necessary in the near future. The increased storage capacity, media retention and computer usage of the general population will feed the increased trends reported by the FBI of digital media becoming relevant in investigations. The high cost and lengthy training required for expert digital forensic investigators has created a backlog of digital media to be analyzed within organizations with budget limitations. For the typical investigator, an automated report on media associated to a case that provides basic information about the media

and data stored within can provide a triage capability to focus the limited man hours of the expert digital forensic investigator available for case analysis. By automating the digital forensic process from data ingestion through reporting, and returning that report to the investigator to decide whether further analysis is merited or required, the investigator is freed to focus on other case aspects and the trained digital investigators can focus on significant tasks.

8.2 Future Work

There is much work that must be done to make this automated reporting vision a productive reality. PyFlag requires fundamental redesign in the database and report display areas. The current database seems to be developed ad-hoc as new functionality was added to the system. There exists no means to provide cross case indexing or identity resolution, which can yield productive results when investigating large organizations and networks. These capabilities can direct the investigator towards previously unknown communication nodes.

PyFlag needs more data extractors developed to increase usefulness. Many communications protocols are not implemented in the current extraction capability, nor are analysis of application storage implementations. Also, analysis of non computer related media such as cell phones will eventually be required so that call records, phone books and SMS messages can be analyzed and possibly resolved with other communications relevant to the investigation.

The PyFlag database schema requires a fundamental redesign with inter-case correlation and identity resolution capabilities in mind. Communications methods should be abstracted and stored at the most basic level in the database to facilitate faster querying of multiple communications streams. Unique identifying information should be cross indexed between cases to provide instant correlation to other relevant information.

The proposed tool plug-in should allow for regression testing against known data sets to ensure proper operation, though defining test data for unknown tool capabilities presents a larger problem beyond the scope of this thesis. Finally, identity resolution capability should be integrated so that varied and incomplete data sources can provide a better “big picture” analysis of an investigation.

The task of choosing what is considered relevant data and what is not relevant was not discussed in depth and in this thesis, this task will necessarily vary from case to case and with the technical skill of the suspect. But there are basic pieces of relevant information - for example, a disk's

users and their online electronic identities that are sure to be relevant to the vast majority of investigations. By focusing on such data, significant improvements can be made in the field of automated Digital Forensic reporting.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:

Sample Conceptual Report

The following pages present a conceptual report created in Microsoft Word to demonstrate the type of information that we desire to provide to the investigator through this automated concept.

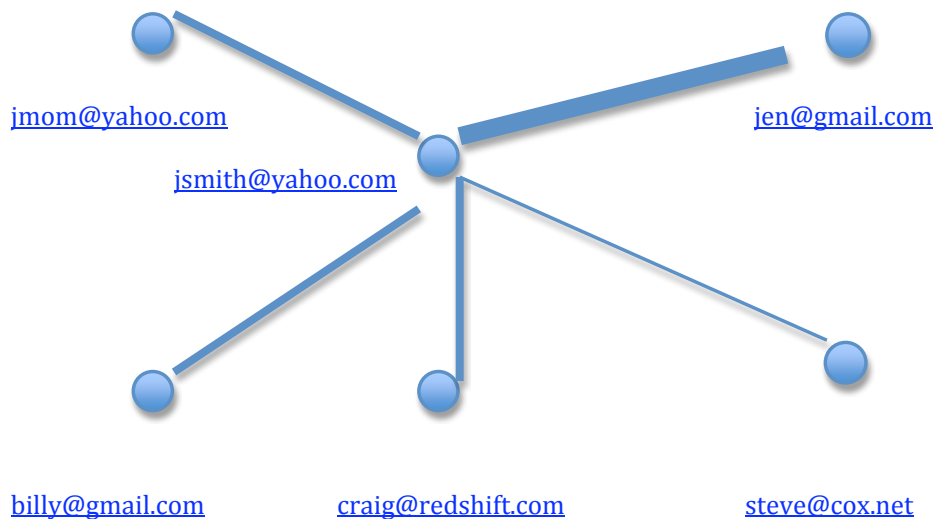
Report for drive "July2008_Smith_Investigation" generated 22JUL08

User accounts	Last Login	Full Name
Administrator	15JUN08	Administrator
Guest	Never	Guest
Jsmith	21JUL08	John Smith

Signature Matches:

Stegprogram.exe	SHA-1 match	Suspect_Programs.blm	Jsmith
SecureErase.exe	SHA-1 match	Suspect Programs.blm	Jsmith
NISTSecret.pdf	SHA-1 match	FilesPossiblyStolen.blm	Jsmith
NISTEmails.doc	SHA-1 match	FilesPossiblyStolen.blm	Jsmith

Email account	User	Sent	Received	Most Recent
jsmith@yahoo.com	jsmith	261	692	21JUL08
Top 5 Destinations				
jmom@yahoo.com	mom	20	21	21JUL08
billy@gmail.com	bill fitz	32	15	18JUL08
steve@cox.net	steve jones	21	12	10FEB08
craig@redshift.com	craig collins	15	31	28JUN08
jen@gmail.com	jen oliver	68	74	21JUL08



Email account	User	Sent	Received	Most Recent
secretj@yahoo.com	jsmith	38	54	20JUL08
Top 5 Destinations				
durka@yahoo.com	durk	28	30	19JUL08

aarnold@nist.gov

aaron arnold 20 24

10JUL08



durka@yahoo.com

secretj@yahoo.com

aarnold@nist.gov

Web Browser Internet Explorer

Bookmarks

<http://mail.yahoo.com>

<http://ebay.com>

<http://www.bostonnews.com>

<http://www.someforumsite.com>

Last 10 Sites

www.yahoo.com

www.cnn.com

www.someforumsite.com

www.mail.yahoo.com

www.cnn.com

www.bostonnews.com

www.etc.com

www.flickr.com

Date

21JUL08

21JUL08

20JUL08

21JUL08

21JUL08

17JUL08

18JUL08

20JUL08

Histogram

500 Hits

330 Hits

25 Hits

400 Hits

330 Hits

55 Hits

15 Hits

43 Hits

Site Credentials (Cookies)

<http://mail.yahoo.com> username: jsmith

<http://mail.yahoo.com> username: secretj

<http://forum.someforumsite.com> username: secretj

Images:



Documents:

Microsoft Word: 73 Documents Found 1 File Encrypted
72 Files Created by Computer Owner
1 File Not Created by Computer Owner:
NISTEmails.doc Created by: A. Durk.

Adobe PDF: 5 Documents Found 0 Files Encrypted
4 Documents found via google.com search
1 File not found via search:
NISTSecret.pdf

Instant Messenger Logs:

Aim.exe Found: AOL Instant Messenger Account: jsmith
No Logs Found

Pidgen.exe Found: AOL Instant Messenger Account jsmith
Logs Found:
BillySoccerPlayer 10JUL08 to 21JUL08 635 lines
Chris242 11JUL08 to 20JUL08 210 lines
Jmomonline 10JUL08 to 15JUL08 58 lines

Pidgen.exe Found: Yahoo Chat Account secretj
Logs Found:
Durka 15JUL08 to 21JUL08 937 lines
Aarnold 15JUL08 to 21JUL08 412 lines

APPENDIX B:

Actual Generated Report

Following is a proof of concept report run against the PyFlag case database after ingestion and analysis of the realistic disk image. Several parts of the database are not populated by the current versions of PyFlag. When trying to load the NSRL RDS database into PyFlag version 0.87, the `nsrl_load.py` utility returns SQL errors on unsupported format characters. PyFlag also does not currently populate the windows registry database tables. As PyFlag matures, the more populated database will yield more information to these reports. The report also contains code to generate IM information if available.

Report on realistic1 Case

Signature Matches

File Name	Hash	Product Name	File Date
default	-		2008-10-30 09:50:31
SAM	-		2008-10-30 09:50:31
SECURITY	-		2008-10-30 09:50:31
software	-		2008-10-30 09:50:31
system	-		2008-10-30 09:50:31
NTUSER.DAT	-		2008-10-30 09:50:31
UsrClass.dat	-		2008-10-30 09:50:31
NTUSER.DAT	-		2008-10-30 09:50:31
UsrClass.dat	-		2008-10-30 09:50:31
wuapi.dll	-		2008-10-30 09:50:18

user 'domex1'

user: domex1

Most Popular Files

Filetype	Count
	1511
.gif	160
.js	140
.properties	96
.xml	88
.jpg	61
.dtd	48
.htm	42
.rdf	39
.ini	32
.txt	29

Most Recent Files

Filename	Time
NTUSER.DAT	2008-10-30 09:47:55

ntuser.ini	2008-10-30 09:47:55
UsrClass.dat	2008-10-30 09:47:55
Application Data	2008-10-30 09:47:45
NTUSER.DAT.LOG	2008-10-30 09:47:37
common.cls	2008-10-30 09:47:08
cert8.db	2008-10-30 09:47:07
key3.db	2008-10-30 09:47:07
Temp	2008-10-30 09:47:02
6.8.12.4	2008-10-30 09:47:02

Most Popular Websites

Address	Count
static.cache.l.google.com	165
www.aolcdn.com	79
mail.google.com	59
co101w.col101.mail.live.com	54
webmail.mozdev.org	50
bl120w.bl120.mail.live.com	50
cache.lifehacker.com	42
tk2.stc.s-msn.com	25
www.google.com	22
tk2.stb.s-msn.com	20
gfx1.hotmail.com	17

Images

File Link	File Name	File Size
None	Windmills426637736[1].jpg	83923
None	Gondolas1202795864[1].jpg	83919
None	32769399d01	72843
None	hsm3promo[1].jpg	64504
None	29C1EC8Dd01	64504
None	13FFEC6Dd01	63040
None	48d3b9a1-0022a-06e32-400cb8e1[1].jpg	61210
None	48fceb6d-0008d-0637f-400cb8e1[1].jpg	60002
None	F79D2850d01	59698
None	sexychat[1].jpg	58887
None	343408E0d01	58887
None	48fcd482-003cb-0637f-400cb8e1[1].jpg	56753
None	pixnay16[1].jpg	49916

None	B9900074d01	49916
None	5AE35B40d01	48840
None	48dd2074-002b3-07303-400cb8e1[1].jpg	47901
None	9FCDB21Bd01	45851
None	aimtoolbar150[1].jpg	45851
None	AE71DA23d01	45802
None	3997BF36d01	42420

Microsoft Office Files

File Link File Name File Size

None excel4.xls 1518

user 'domex2'

user: domex2

Most Popular Files

Filetype Count

	462
.gif	157
.htm	66
.jpg	60
.js	37
.ini	33
.txt	27
.css	24
.dat	23
.lnk	22
.xml	20

Most Recent Files

Filename	Time
UsrClass.dat	2008-10-30 09:49:00
NTUSER.DAT	2008-10-30 09:43:20
NTUSER.DAT.LOG	2008-10-30 09:43:20
Recent	2008-10-30 00:50:17
Local Settings	2008-10-30 00:50:13
My Documents	2008-10-30 00:50:13

Start Menu	2008-10-30 00:50:13
Application Data	2008-10-30 00:50:13
My Music	2008-10-30 00:50:13
My Pictures	2008-10-30 00:50:13

Most Popular Websites

Address	Count
bl126w.blu126.mail.live.com	23
bl135w.blu135.mail.live.com	17
gfx1.hotmail.com	11
gfx2.hotmail.com	10
login.live.com	8
m1.2mdn.net	7
gfx8.hotmail.com	7
www.google.com	6
h.msn.com	6
gfx6.hotmail.com	6
rad.live.com	5

Images

File Link	File Name	File Size
None	4FFD0FE4d01	30199
None	mAtL_inbox[1].jpg	27894
None	gmail_77659b_en[1].gif	27006
None	gmail_77659c_en[1].gif	25526
None	32[1].png	23940
None	skydrive_overview_2[1].jpg	23556
None	Settings[2].jpg	23259
None	toolbar2[1].jpg	22860
None	g_overview[1].jpg	22566
None	Events_SuitePage_Graphic.cropped8[1].gif	22468
None	maildesk2[1].jpg	21109
None	D9A13DFBd01	20822
None	5D53F2EEd01	20264

Microsoft Office Files

File Link	File Name	File Size
None	excel4.xls	1518

user 'Administrator'

user: Administrator

Most Popular Files

Filetype Count

	593
.gif	214
.js	198
.png	164
.dll	147
.htm	94
.jpg	79
.css	62
.txt	37
.ini	32
.cab	32

Most Recent Files

Filename	Time
NTUSER.DAT	2008-10-30 09:50:00
ntuser.ini	2008-10-30 09:50:00
UstrClass.dat	2008-10-30 09:50:00
ntuser.dat.LOG	2008-10-30 09:49:56
IconCache.db	2008-10-30 09:49:49
Preferred	2008-10-30 09:10:59
goopdateres_en.dll	2008-10-30 09:10:59
Local Settings	2008-10-30 08:56:40
Update	2008-10-30 08:56:40
1.2.131.25	2008-10-30 08:56:40

Most Popular Websites

Address	Count
help.live.com	21
static.cache.l.google.com	19
co101w.col101.mail.live.com	18
bl120w.blu120.mail.live.com	18

shared.live.com	13
login.live.com	10
gfx1.hotmail.com	10
gfx2.hotmail.com	8
m1.2mdn.net	6
gfx8.hotmail.com	6
gfx6.hotmail.com	6

Images

File Link	File Name	File Size
None	promoimage1[1].jpg	117956
None	100308_aol[1].jpg	99476
None	GreenBizLogo_72ppi[1].jpg	84940
None	dlpage_lg[1].jpg	76002
None	background-firefox-1[1].jpg	73350
None	f_000002	71038
None	f_000003	70424
None	ss2[1].jpg	68044
None	f_000001	64187
None	feature-background-2[1].jpg	63296
None	background-thunderbird-features[1].jpg	49930
None	Install_AIM[1].exe	49930
None	chrome-205_noshadow[1].png	49564
None	background-download[1].jpg	45022
None	splash[1].png	36767
None	omni-2_300x250[1].jpg	36541
None	aimani.gif	34127
None	purple_logo_w_text[1].jpg	31112
None	promo_bg14[1].jpg	30718
None	promo_bg68b[1].jpg	30043

Microsoft Office Files

File Link	File Name	File Size
None	excel4.xls	1518

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C:

Sample Adium Log

```
<?xml version="1.0" encoding="UTF-8" ?>
<chat xmlns="http://purl.org/net/ulf/ns/0.4-02" account="domextest1" service="AIM">
<event type="windowOpened" sender="domextest1" time="2008-09-18T22:06:06-07:00"/>
<message sender="domextest2" time="2008-09-18T22:06:06-07:00">
  <div><span style="font-family: Helvetica; font-size: 12pt;">test</span>
</div></message>
<message sender="domextest1" time="2008-09-18T22:06:07-07:00" alias="p f">
  <div><span style="font-family: Helvetica; font-size: 12pt;">test</span>
</div></message>
<message sender="domextest1" time="2008-09-18T22:06:15-07:00" alias="p f">
  <div><span style="font-family: Helvetica; font-size: 12pt;">
    this is a message from adium to pidgin</span></div></message>
<message sender="domextest2" time="2008-09-18T22:06:22-07:00"><div>
  <span style="font-family: Helvetica; font-size: 12pt;">
    this is a message from pidgin to adium</span></div></message>
<event type="windowClosed" sender="domextest1" time="2008-09-18T22:06:35-07:00"/>
</chat>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D:

Sample Pidgin Log

Conversation with domextest1 at Thu 18 Sep 2008 09:50:28 PM PDT on domextest2 (aim)

(09:50:30 PM) domextest2: test

(09:50:41 PM) domextest1: test

(09:50:52 PM) domextest1: this is a message from aim6 to pidgin

(09:50:59 PM) domextest2: this is a message from pidgin to aim6

(10:06:24 PM) domextest2: test

(10:06:26 PM) domextest1: test

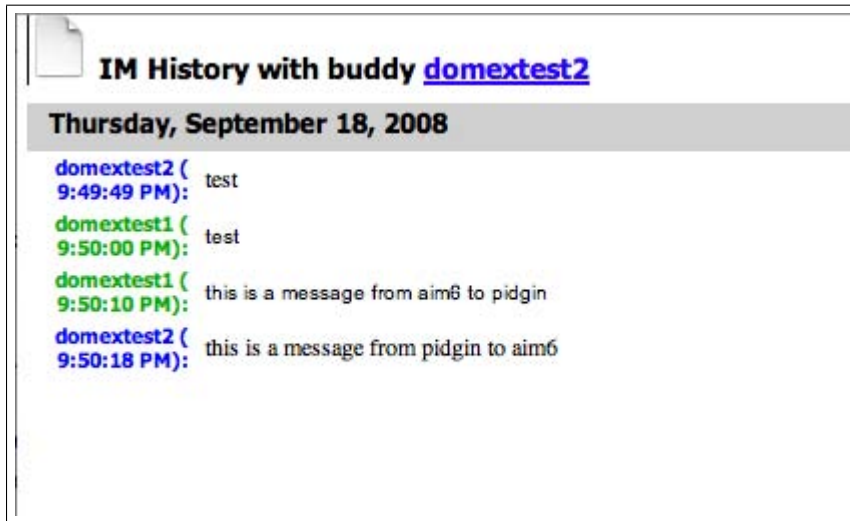
(10:06:33 PM) domextest1: this is a message from adium to pidgin

(10:06:40 PM) domextest2: this is a message from pidgin to adium

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E:

Sample AOL Instant Messenger Log



```
<?xml version="1.0" standalone="yes" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="content-type" content="application/xhtml+xml; charset=utf-8"/>
<title>IM History with buddy dometest2</title>
<link rel="stylesheet" href="styles.css" type="text/css">
</head>
<body><h1>
&#160;IM History with buddy
<a href="aim:goim?screenName=dometest2&targetBuddyList=dometest1">
dometest2</a></h1>
<table width="100%" cellpadding="1" cellspacing="0">
<tr><td colspan="2" class="time">Thursday, September 18, 2008</td></tr>
<tr><td class="remote">dometest2&#160;(9:49:49&#160;PM) :</td><td class="msg"
width="100%">test</td></tr>
<tr><td class="local">dometest1&#160;(9:50:00&#160;PM) :</td><td class="msg"
width="100%">
<font face="Arial" size="2" color="#000000">test</font></td></tr>
```

```

<tr><td class="local">domextest1&#160;(9:50:10&#160;PM):</td><td class="msg"
width="100%">
  <font face="Arial" size="2" color="#000000">
    this is a message from aim6 to pidgin</font>
  </td></tr>
<tr><td class="remote">domextest2&#160;(9:50:18&#160;PM):</td>
  <td class="msg" width="100%">this is a message from pidgin to aim6</td></tr>
</table>
<a name="end"></a>
</body></html>

```

APPENDIX F:

userreport.py

```
#!/usr/bin/python
# Examining the PyFlag database and report on a case

import time,sys,os,os.path,re
from sets import Set
import MySQLdb
import domex

bogus_users = ["NetworkService","All Users","Default User","SMSCCMBootAcct&","SMSCliToknAcct&","SMSCliToknLocalAcct&","SMSCliSvcAcct&","LocalService"]

def users(c):
    files = c.execute("select path from file where path like '%%Documents and Settings%%'")
    res = c.fetchall()

    multiuser_re = re.compile("([^\s]+)/Documents and Settings/([^\s]+)/")
    from sets import Set
    users = Set()
    for (file,) in res:
        m = multiuser_re.search(file)
        if m:
            if m.group(2) not in bogus_users:
                users.add((m.group(1),m.group(2)))
    print "<h1>Files by user</h1>"

    for (vfs,user) in users:
        print "<h2>user '%s'</h2>\n" % user
        print "user:",user
        c.execute("""select name from file where path like '/'+"vfs+"/Documents and Settings/"+user+"%%'""")
        res = c.fetchall()
        hist = {}
        for (file,) in res:
            pos = file.rfind(".")
            if pos== -1:
                filetype = "<none>"
            else:
                filetype = file[pos:]
            hist[filetype] = hist.get(filetype,0) + 1
        rev = [(v,k) for k,v in hist.items()]
        rev.sort()
        rev.reverse()
        hist = [(v,k) for k,v in rev]

        print "<table>"
        print "<tr><th>Filetype</th><th>Count</th></tr>\n"
        i=0;
        for key,count in hist:
            print "<tr><td>%s</td><td>%d</td></tr>\n" % (key,count)
            i+=1
            if i>10:
                break
        print "</table>\n"
        webhistory(c,user)

def signatures(c):
    files = c.execute("select file.name, hash.NSRL_product, inode.atime from inode join file on file.inode_id = inode.inode_id join hash on ha
    res = c.fetchall()
    print "<h1>Signature Matches</h1>\n"
    print "<table>"
    print "<tr><th>File Name</th><th>Hash Product Name</th><th>File Date</th></tr>\n"
    for (name, hash, time) in res:
```

```

        print "<tr><td>%s</td><td>%s</td><td>%s</td></tr>\n" % (name, hash, time)
    print "</table>\n"

def emails(c):
    files = c.execute("select email.from, email.to, email.date, email.subject from email")
    res = c.fetchall()
    subjs = {}
    sends = {}
    for (efrom, eto, edate, esubj) in res:
        subjs[esubj] = subjs.get(esubj,0)+1
        sends[efrom] = sends.get(efrom,0)+1
    rev = [(v,k) for k,v in sends.items()]
    rev.sort()
    rev.reverse()
    sends = [(v,k) for k,v in rev]
    rev = [(v,k) for k,v in subjs.items()]
    rev.sort()
    rev.reverse()
    subjc = [(v,k) for k,v in rev]
    print "<h1>Most Popular Email Subjects</h1>\n"
    print "<table>"
    print "<tr><th>Subject</th><th>Count</th></tr>\n"
    i=0;
    for key,val in subjs:
        print "<tr><td>%s</td><td>%d</td></tr>\n" % (key,val)
        i+=1
        if i>10:
            break
    print "</table>\n"
    print "<h1>Most Popular Email Senders</h1>\n"
    print "<table>"
    print "<tr><th>Subject</th><th>Count</th></tr>\n"
    i=0;
    for key,val in sends:
        print "<tr><td>%s</td><td>%d</td></tr>\n" % (key,val)
        i+=1
        if i>10:
            break
    print "</table>\n"

def webhistory(c,u):
    files = c.execute("select http.url, file.path from http, inode, file where http.inode_id = inode.inode_id and inode.inode_id = file.inode_id")
    res = c.fetchall()
    address_re = re.compile("http:\\/\\/[^\/]*/[.]*")
    sites = {}
    for (url,) in res:
        s=address_re.search(url)
        if s:
            s=s.group(1)
            sites[s] = sites.get(s,0)+1
    rev = [(k,v) for v,k in sites.items()]
    rev.sort()
    rev.reverse()
    sites = [(v,k) for k,v in rev]
    print "<h1>Most Popular Websites</h1>\n"
    print "<table>"
    print "<tr><th>Address</th><th>Count</th></tr>\n"
    i=0;
    for key,val in sites:
        print "<tr><td>%s</td><td>%d</td></tr>\n" % (key,val)
        i+=1
        if i>10:
            break
    print "</table>\n"

def images(c,u):
    files = c.execute("select file.link, file.name, inode.size from inode join 'file' on 'inode'.inode_id = 'file'.inode_id join 'type' on 'ir")
    res = c.fetchall()
    print "<h1>Images</h1>\n"
    print "<table>"

```

```

print "<tr><th>File Link</th><th>File Name</th><th>File Size</th></tr>\n"
for (name, size, time) in res:
    print "<tr><td>%s</td><td>%s</td><td>%s</td></tr>\n" % (name, size, time)
print "</table>\n"

def documents(c,u):
    files = c.execute("select file.link, file.name, inode.size from inode join 'file' on 'inode'.inode_id = 'file'.inode_id join 'type' on 'in")
    res = c.fetchall()
    print "<h1>Microsoft Office Files</h1>\n"
    print "<table>"
    print "<tr><th>File Link</th><th>File Name</th><th>File Size</th></tr>\n"
    for (name, size, time) in res:
        print "<tr><td>%s</td><td>%s</td><td>%s</td></tr>\n" % (name, size, time)
    print "</table>\n"

def im(c,u):
    files = c.execute("select file.name, inode.size, inode.atime from file, inode, type where file.inode_id = inode.inode_id and inode.inode_id")
    res = c.fetchall()
    print "<h1>Instant Messenger Logs</h1>\n"
    print "<table>"
    print "<tr><th>File Name</th><th>File Size</th><th>Date</th></tr>\n"
    for (name, size, time) in res:
        print "<tr><td>%s</td><td>%s</td><td>%s</td></tr>\n" % (name, size, time)
    print "</table>\n"
    files = c.execute("select nick, COUNT(*) as count from msn_users order by count desc limit 10")
    res = c.fetchall()
    print "<h1>Instant Messenger Conversations</h1>\n"
    print "<table>"
    print "<tr><th>Nickname</th><th>Count</th></tr>\n"
    for (name, count) in res:
        if count > 0:
            print "<tr><td>%s</td><td>%s</td></tr>\n" % (name, count)
    print "</table>\n"

if (__name__=='__main__'):
    from optparse import OptionParser
    parser = OptionParser()
    parser.usage = "usage: %prog [options] <database>\n\nReports on a PyFlag case"
    (options,args) = parser.parse_args()
    mysql = domex.mysql_connection(db=sys.argv[1])
    c = mysql.cursor()

    print "<html><head><title>%s User Report </title></head>" % sys.argv[1]
    print "<body>"
    print "<h1>Report on %s Case</h1>\n" % sys.argv[1]
    signatures(c)
    users(c)
    print "</body></html>\n"

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G:

domex.py

```
""" The python domex.py library. Contains all site-wide definitions for our domex system """

import os

def mysql_connection(db):
    """Returns a MySQL connection to the DOMEX SQL database"""
    for f in ['/var/run/mysqld/mysqld.sock', '/var/mysql/mysql.sock', '/tmp/mysql.sock']:
        if os.path.exists(f):
            sock = f
            break
    import MySQLdb
    """if not db: db=os.getenv("DOMEX_MYSQL_DATABASE")"""

    mysql = MySQLdb.connect(host="127.0.0.1",
                            user="root",
                            passwd="password",
                            db=db,
                            unix_socket=sock)

    return mysql

def get_aff_files():
    """Return a list of the AFF files on the server"""
    ret = []
    for (dirpath, dirnames, filenames) in os.walk(os.getenv("DOMEX_CORP")):
        for filename in filenames:
            if filename.endswith(".aff"):
                ret.append(dirpath + filename)
    return ret

def get_xml_files():
    """Return a list of the AFF files on the server"""
    ret = []
    for (dirpath, dirnames, filenames) in os.walk(os.getenv("DOMEX_CORP")):
        for filename in filenames:
            if filename.endswith(".xml"):
                ret.append(dirpath + "/" + filename)
    return ret

def find_aff(basename):
    for (dirpath, dirnames, filenames) in os.walk(os.getenv("DOMEX_CORP")):
        for filename in filenames:
            if basename==filename: return dirpath + "/" + filename;
    return None

def affgid(fn):
    from subprocess import Popen, PIPE
    return Popen(['afsegment', '-pimage_gid', fn], stdout=PIPE).stdout.read()

def aff_gid_hex(fn):
    import hex
    return hex.bin2hex(affgid(fn))
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H:

IMLogMagic.py

```
""" This file contains magic classes to identify files from their
headers more accurately
"""
import pyflag.Magic as Magic
import pyflag.Registry as Registry
import pyflag.DB as DB
import pyflag.FlagFramework as FlagFramework

class PidginLogMagic(Magic.Magic):
    type = "Instant Messenger Log File"
    mime = "text/plain"
    default_score = 100

    regex_rules = [
        ( "Conversation with [\w]+ at [\w\w\w] [\d]+ [\w\w\w] [\d]{4} [\w\w \w\w\w] on [\w]+ \([{\w}+\\)\n", (0,100)),
        ( "\([{\d\d}\:]{\d\d}\:){\d\d} [\w]+\: [\w\s]+\n", (0,1000))
    ]

    samples = [ (100, "Conversation with domextest1 at Thu 18 Sep 2008 09:50:28 PM PDT on domextest2 (aim)")]

class AIMLogMagic(Magic.Magic):
    type = "Instant Messenger Log File"
    mime = "text/plain"
    default_score = 100

    regex_rules = [
        ( "<title>IM History with buddy [\w]+</title><link rel=\"stylesheet\" href=\"styles.css\" type=\"text/css\">", (0,100)),
        ( "<tr><td class=\"local\">[\w]+&#160;(\d:\d\d:\d\d&#160;\w\w):", (0,1000)),
        ( "<tr><td class=\"remote\">[\w]+&#160;(\d:\d\d:\d\d&#160;\w\w):", (0,1000))
    ]

    samples = [ (100, ""<tr><td class="local">domextest1&#160;(9:50:00&#160;PM):</td><td class="msg" width="100%"><font face="Arial" size="2" col

class AdiumLogMagic(Magic.Magic):
    type = "Instant Messenger Log File"
    mime = "text/plain"
    default_score = 100

    regex_rules = [
        ( "<chat xmlns=\"http://purl.org/net/ulf/ns/0.4-02\" account=", (0,100)),
        ( "<message sender=", (0,1000))
    ]

    samples = [ (100, ""<message sender="domextest2" time="2008-09-18T22:06:06-07:00"><div><span style="font-family: Helvetica; font-size: 12pt;

class AIMLogScanner(Scanner.GenScanFactory):
    """Scan AIM Logs to insert data into msn_users table"""
    depends = 'TypeScan'
    group = 'FileScanners'

    def __init__(self, fsfd):
        Scanner.GenScanFactory.__init__(self,fsfd)
        dbh=DB.DBO(self.case)

    class Scan(Scanner.StoreAndScanType):
        types = ('Instant Messenger Log File')

        def external_process(self, fd):
            if self.mime_type==self.types[0]:
                self.process_log(fd)
```

```

def process_log(self, fd):
    try:

        path, inode, indoe_id = self.ddfs.lookup(inode=fd.inode)
        line = fd.readline()
        local_user = ""
        remote_user = ""
        message_text = ""
        if line.startswith('<body><h1><img src='):
            info_re = re.compile("<body><h1><img src=\"http://api.oscar.aol.com/expressions/get?f=native&type=buddy")
            match = info_re.search(line)
            local_user = match.group("local_user")
            remote_user = match.group("remote_user")

        if line.startswith('<tr><td class=\\"local\\>'):
            local_re = re.compile("<tr><td class=\"local\">(P<local_user>\\w+)&#160;(<d:<d:<d:<d:&#160;\\w\\w):</td><td class=\"msg\" w")
            match = local_re.search(line)
            message_text = match.group("message")
            dbh=DB.DBO(self.case)
            dbh.insert('msn_users',
                        inode_id = self.inode,
                        nick = local_user,
                        user_data_type = 'personal_message',
                        user_data = message_text)
            message_text = ""

        if line.startswith('<tr><td class=\\"remote\\>'):
            local_re = re.compile("<tr><td class=\"remote\">(P<remote_user>\\w+)&#160;(<d:<d:<d:<d:&#160;\\w\\w):</td><td class=\"msg\" w")
            match = local_re.search(line)
            message_text = match.group("message")
            dbh.insert('msn_users',
                        inode_id = self.inode,
                        nick = remote_user,
                        user_data_type = 'personal_message',
                        user_data = message_text)
            message_text = ""
    except Exception,e:
        pyflaglog.log(pyflaglog.DEBUG,"AIMLogScanner Scan: Unable to parse inode %s as an AIM log message (%s)" % (self.inode,e))

class AdiumLogScanner(Scanner.GenScanFactory):
    """Scan Adium Logs and insert data into msn_users table"""
    depends = ['TypeScan']
    group = 'FileScanners'

    def __init__(self, fsfd):
        Scanner.GenScanFactory.__init__(self,fsfd)
        dbh=DB.DBO(self.case)

class Scan(Scanner.StoreAndScanType):
    types = ['Instant Messenger Log File']

    def external_process(self, fd):
        if self.mime_type==self.types[0]:
            self.process_log(fd)

    def process_log(self, fd):
        try:

            path, inode, indoe_id = self.ddfs.lookup(inode=fd.inode)
            line = fd.readline()
            local_user = ""
            remote_user = ""
            message_text = ""
            if line.startswith("<chat xmlns=\"http://purl.org/net/ulf/ns/0.4-02\" account=\""):
                info_re = re.compile("<chat xmlns=\"http://purl.org/net/ulf/ns/0.4-02\" account=\"(P<local_user>\\w+)\" s")
                match = info_re.search(line)
                local_user = match.group("local_user")

            if line.startswith("<message sender=\""):

```

```

        local_re = re.compile("""<message sender="(?"<user_name>\w+)" time="\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}-\d{2}:\d{2}>""")
        match = local_re.search(line)
        message_text = match.group("message")
        nickname = match.group("user_name")
        dbh.insert('msn_users',
                    inode_id = self.inode,
                    nick = nickname,
                    user_data_type = 'personal_message',
                    user_data = message_text)
        message_text = ""

    except Exception,e:
        pyflaglog.log(pyflaglog.DEBUG,"AdiumLogScanner Scan: Unable to parse inode %s as an Adium log message (%s)" % (self.inode,e))

class PidginLogScanner(Scanner.GenScanFactory):
    """Scan Pidgin Logs and insert data into msn_users table"""
    depends = ['TypeScan']
    group = 'FileScanners'

    def __init__(self, fsfd):
        Scanner.GenScanFactory.__init__(self,fsfd)
        dbh=DB.DBO(self.case)

class Scan(Scanner.StoreAndScanType):
    types = ['Instant Messenger Log File']

    def external_process(self, fd):
        if self.mime_type==self.types[0]:
            self.process_log(fd)

    def process_log(self, fd):
        try:

            path, inode, indoe_id = self.ddfs.lookup(inode=fd.inode)
            line = fd.readline()
            local_user = ""
            remote_user = ""
            message_text = ""
            if line.startswith("""Conversation with """):
                info_re = re.compile("""Conversation with (?<remote_user>\w+) at \w{3} \d{2} \w{3} \d{4} \d{2}:\d{2}:\d{2}>""")
                match = info_re.search(line)
                local_user = match.group("local_user")
                remote_user = match.group("remote_user")

            if line.startswith("""(\d{2}:\d{2}:\d{2} \w{2}) """):
                local_re = re.compile("""(\d{2}:\d{2}:\d{2} \w{2}) (?<username>\w+): (?<message>[\w\W]+) """)
                match = local_re.search(line)
                message_text = match.group("message")
                nickname = match.group("user_name")
                dbh.insert('msn_users',
                            inode_id = self.inode,
                            nick = nickname,
                            user_data_type = 'personal_message',
                            user_data = message_text)
                message_text = ""

        except Exception,e:
            pyflaglog.log(pyflaglog.DEBUG,"PidginLogScanner Scan: Unable to parse inode %s as an Pidgin log message (%s)" % (self.inode,e))

```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

- [1] Forensic toolkit 2.0, 2008. <http://www.accessdata.com/Products/ftk2test.aspx>. [Online; accessed 11 June 2008].
- [2] Robert-Jan Mora Bas Kloet, Joachim Metz. libewf: Project info, May 2008. <http://www.uitwisselplatform.nl/projects/libewf/>. [Online; accessed 6 June 2008].
- [3] Steven Bassi. An automated acquisition system for media exploitation. Master's thesis, Naval Postgraduate School, June 2008.
- [4] Brian Carrier. Autopsy forensic browser: Description. <http://www.sleuthkit.org/autopsy/desc.php>. [Online; accessed 03 January 2009].
- [5] Brian Carrier. The Sleuth Kit & Autopsy: Forensics tools for Linux and other Unixes, 2005. <http://www.sleuthkit.org/>. [Online; accessed 06 March 2009].
- [6] Brian Carrier. *A Hypothesis-Based Approach to Digital Forensic Investigations*. PhD thesis, Purdue University, 2006.
- [7] M.I. Cohen. Pyflag: An advanced network forensic framework. In *Proceedings of the 2008 Digital Forensics Research Workshop*. DFRWS, August 2008. <http://www.pyflag.net>. [Online; accessed 06 March 2009].
- [8] Michael I. Cohen. Advanced jpeg carving. In *e-Forensics '08: Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop*, pages 1–6. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008. ISBN 978-963-9799-19-6.
- [9] Cray Inc. Cray inc., the supercomputer company - about cray - history. <http://www.cray.com/About/History.aspx>. [Online; accessed 20 February 2009].
- [10] Ian F. Darwin. Libmagic, August 2008. <ftp://ftp.astron.com/pub/file/>.
- [11] DF Labs. Ptk overview. <http://ptk.dflabs.com/overview.html>. [Online; accessed 03 January 2009].

- [12] Digital Assembly, LLC. Why adroit photo recovery? <http://digital-assembly.com/products/>. [Online; accessed 02 March 2009].
- [13] Digital Forensic Research Workshop. A roadmap for digital forensic research, 2001. <http://dfrws.org/2001/dfrws-rm-final.pdf>. [Online; accessed 03 September 2008].
- [14] Digital Forensic Research Workshop. Dfrws 2008 forensics challenge results, 2008. <http://dfrws.org/2008/challenge/results.shtml>. [Online; accessed 03 January 2009].
- [15] Dan Farmer and Wietse Venema. The coroner's toolkit (tct). <http://www.porcupine.org/forensics/tct.html>. [Online; accessed 02 March 2009].
- [16] Paul Farrell, Simson Garfinkel, and Doug White. Practical applications of bloom filters to the nist rds and hard drive triage. In *Annual Computer Security Applications Conference 2008*, December 2008.
- [17] Federal Bureau of Investigation. Regional computer forensic laboratory annual report 2007. http://www.rcfl.gov/downloads/documents/RCFL_Nat_Annual07.pdf. [Online; accessed 06 March 2009].
- [18] Simson L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 2007.
- [19] Simson L. Garfinkel, David J. Malan, Karl-Alexander Dubec, Christopher C. Stevens, and Cecile Pham. Disk imaging with the advanced forensic format, library and tools. In *Research Advances in Digital Forensics (Second Annual IFIP WG 11.9 International Conference on Digital Forensics)*. Springer, January 2006.
- [20] Guidance Software. Guidance software form 10-k, 2007. <http://investors.guidancesoftware.com/secfiling.cfm?filingID=1193125-08-57761>. [Online, accessed 17 June 2008].
- [21] Guidance Software. Encase annual training passport, 2008. <http://www.guidancesoftware.com/training/AnnualTraining.aspx>. [Online; accessed 06 March 2009].

- [22] Guidance Software, Inc. EnCase Forensic, 2007. http://www.guidancesoftware.com/products/ef_index.asp.
- [23] Golden G. Richard III and V. Roussev. Scalpel: A frugal, high performance file carver. In *Proceedings of the 2005 Digital Forensics Research Workshop*. DFRWS, August 2005. <http://www.digitalforensicssolutions.com/Scalpel/>.
- [24] Jeff Jonas. Threat and fraud; intelligence, las vegas style. *IEEE Security and Privacy*, 6: 28–34, 2006.
- [25] Nicholas Mikus. An analysis of disc carving techniques. Master’s thesis, Naval Postgraduate School, March 2005.
- [26] Nick Mikus, Kris Kendall, and Jesse Kornblum. Foremost(1), January 2006. <http://foremost.sourceforge.net/foremost.html>. [Online; accessed 06 March 2009].
- [27] National Institute for Standards and Technology. Federal information processing standards publication 180-1, 1995.
- [28] Cfft project overview. http://www.cfft.nist.gov/project_overview.htm, National Institute of Standards and Technology. [Online; accessed 28 September 2008].
- [29] National Institute of Standards and Technology. National software reference library reference data set, 2005. <http://www.nsrl.nist.gov/>. [Online; accessed 06 March 2009].
- [30] Anandabrata Pal, Husrev Sencar, and Nasir Memon. Detecting file fragmentation point using sequential hypothesis testing. In *Digital Forensic Research Workshop*, 2008.
- [31] Vassil Roussev, Golden G. Richard III, and Lodovico Marziale. Multi-resolution similarity hashing. *Digital Investigation*, 4(Supplement-1):105–113, 2007.
- [32] Sourcefire Inc. Clam antivirus. <http://www.clamav.net/about/>. [Online; accessed 03 January 2009].
- [33] Bret Swanson. The coming exaflood-bret swanson. *The Wall Street Journal*, 2(26), 2007.
- [34] U.S. Census Bureau. Computer and internet use in the united states: 2003, 2005.

- [35] U.S. Department of Justice. A forensic examination of digital evidence: A guide for law enforcement, April 2004.
- [36] Hal R. Varian and Peter Lyman, 2003. <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/> fi. [Online; accessed 28 September 2008].
- [37] G. Alan Wang, Siddharth Kaza, Shailesh Joshi, Kris Chang, Chunju Tseng, Homa Atabakhsh, and Hsinchun Chen. The arizona idmatcher: developing an identity matching tool for law enforcement. In *dg.o '07: Proceedings of the 8th annual international conference on Digital government research*, pages 304–305. Digital Government Society of North America, 2007. ISBN 1-59593-599-1.
- [38] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd, August 2004. <http://eprint.iacr.org/2004/199.pdf>. [Online; accessed 06 March 2009].

Referenced Authors

Access Data 7	Garfinkel, Simson 30, 47	Pal, Anandabrata 6
Atabakhsh, Homa 6	Garfinkel, Simson L. 5, 7	Pham, Cecile 7
Bas Kloet, Robert-Jan Mora, Joachim Metz 7	Guidance Software 7	Rivest, R. 5
Bassi, Steven 27	Guidance Software, Inc. 7	Roussev, V. 5, 8, 29
	III, Golden G. Richard 5, 8, 29	Roussev, Vassil 5
Carrier, Brian 3, 7	Jonas, Jeff 6	Sencar, Husrev 6
Chang, Kris 6	Joshi, Shailesh 6	Sourcefire Inc. 30
Chen, Hsinchun 6	Kaza, Siddharth 6	Stevens, Christopher C. 7
Cohen, M.I. 8	Kendall, Kris 8	Swanson, Bret 1
Cohen, Michael I. 6	Kornblum, Jesse 8	Technology 5
Cray Inc. 1		Tseng, Chunju 6
Darwin, Ian F. 29	Lai, Xuejia 5	
DF Labs 7	Lyman, Peter 1	U.S. Census Bureau 1
Digital Assembly, LLC. 8	Malan, David J. 7	U.S. Department of Justice 3
Digital Forensic Research Workshop 3, 8	Marziale, Lodovico 5	
Dubec, Karl-Alexander 7	Memon, Nasir 6	Varian, Hal R. 1
	Mikus, Nicholas 5	Venema, Wietse 7
Farmer, Dan 7	Mikus, Nick 8	
Farrell, Paul 30, 47	National Institute for Standards and Technology 5	Wang, G. Alan 6
Federal Bureau of Investigation 1		Wang, Xiaoyun 5
Feng, Dengguo 5	of Standards, National Institute 5	White, Doug 30, 47
		Yu, Hongbo 5

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California